Mining the Meaningful Term Conjunctions from Materialised Faceted Taxonomies: Algorithms and Complexity

Yannis Tzitzikas[1,2] and Anastasia Analyti[2]

[1] *Department of Computer Science, University of Crete, Greece*
[2] *Institute of Computer Science, FORTH-ICS, Crete, Greece*
tzitzik@ics.forth.gr, analyti@ics.forth.gr

Abstract. A *materialized faceted taxonomy* is an information source where the objects of interest are indexed according to a *faceted taxonomy*. This paper shows how from a materialized faceted taxonomy, we can *mine* an expression of the *Compound Term Composition Algebra* that specifies exactly those compound terms (conjunctions of terms) that have non-empty interpretation. The mined expressions can be used for encoding *in a very compact form* (and subsequently reusing), the domain knowledge that is stored in existing materialized faceted taxonomies. A distinctive characteristic of this mining task is that the focus is given on minimizing the storage space requirements of the mined set of compound terms. This paper formulates the problem of expression mining, gives several algorithms for expression mining, analyzes their computational complexity, provides techniques for optimization, and discusses several novel applications that now become possible.
Keywords : Materialized faceted taxonomies, knowledge extraction and reuse, algorithms, complexity

1 Introduction

Assume that we want to build a Catalog of hotel Web pages and suppose that we want to provide access to these pages according to the *Location* of the hotels, the *Sports* that are possible in these hotels, and the *Facilities* they offer. For doing so, we can design a *faceted taxonomy*, i.e. a set of taxonomies, each describing the domain from a different aspect, or *facet*, like the one shown in Figure 1. Now each object (here Web page) can be indexed using a *compound term*, i.e., a set of terms from the different facets. For example, a hotel in Rethimno providing sea ski and wind-surfing sports can be indexed by assigning to it the compound term $\{Rethimno, SeaSki, Windsurfing\}$. We shall use the term *materialized faceted taxonomy* to refer to a faceted taxonomy accompanied by a set of object indices.

However, one can easily see that several compound terms over this faceted taxonomy are meaningless, in the sense that they cannot be applied to any object of the domain. For instance, we cannot do any winter sport in the Greek islands (Crete and Cefalonia) as they never have enough snow, and we cannot do any sea sport in Olympus because Olympus is a mountain. For the sake of this example, let us also suppose that only in Cefalonia there exists a hotel that has a casino, and that this hotel also offers sea ski and windsurfing sports. According to this assumption, the partition of compound terms to the set of *valid* (meaningful) compound terms and *invalid* (meaningless) compound terms is shown in Table 1.

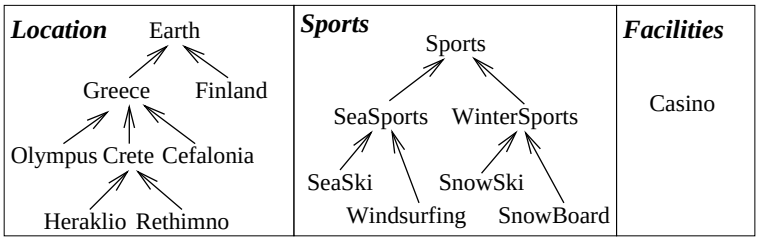

Figure 1: A faceted taxonomy for indexing hotel Web pages

The availability of such a partition would be very useful during the construction of a materialized faceted taxonomy. It could be exploited in the indexing process for preventing indexing errors, i.e. for allowing only meaningful compound terms to be assigned to objects. In particular, knowing this partition, it is possible to generate a "complete" navigation tree, whose dynamically generated nodes correspond to all possible valid compound terms [27]. Such a navigation tree can aid the indexer to select the desired compound term for indexing, by browsing only the meaningful compound terms. This kind of "quality control" or "indexing aid" is especially important in cases where the indexing is done by many people who are not domain experts. For example, the indexing of Web pages in the Open Directory (which is used by Google and several other search engines) is done by more than 20.000 volunteer human editors (indexers). Apart from the indexer, the final user is also aided during his/her navigation and search by browsing only the meaningful compound terms.

However, even from this toy example, it is obvious that the definition of such a partition would be a formidably laborious task for the designer. Fortunately, the recently emerged *Compound Term Composition Algebra* (CTCA) [27] (which is recalled in Section 3.2) can significantly reduce the required effort. According to that approach, the designer can use an algebraic expression to define the valid (meaningful) compound terms by declaring only a *small* set of valid or invalid compound terms from which other (valid or invalid) compound terms are then inferred. For instance, the partition shown in Table 1, can be defined using the expression:

$$e = (Location \ominus_N Sports) \oplus_P Facilities$$

with the following P and N parameters:

$$\begin{aligned} N &= \{\{Crete, WinterSports\},\ \{Cefalonia, WinterSports\}\} \\ P &= \{\{Cefalonia, SeaSki, Casino\},\ \{Cefalonia, Windsurfing, Casino\}\} \end{aligned}$$

In this paper we study the inverse problem, i.e. how we can derive an algebraic expression e (like the above expression) that specifies exactly those compound terms that are *extensionally valid* (i.e. have non-empty interpretation) in an *existing* materialized faceted taxonomy. This problem, which we shall hereafter call *expression mining* or *expression extraction*, has several applications. For instance, it can be applied to materialized faceted taxonomies (which were not defined using CTCA) in order to encode compactly and subsequently reuse the set of compound terms that are extensionally valid (e.g. the set of valid compound terms in Table 1). For example, suppose that we have in our disposal a very large medical file which stores medical incidents classified according to various aspects (like disease, symptoms, treatment, duration of treatment, patient's age, genre, weight, smoking habits, patient's profession, etc.), each one having a form of a hierarchy. In this scenario, expression mining can be used for extracting in a very *compact* form the set of *all* different combinations that have been recorded so far. Note

that here "mining" does not have the classical statistical nature, i.e. we are not trying to mine statistically justified associations or rules [10]. Instead, from an object base we are trying to mine the associations of the elements of the schema (here of the terms of the faceted taxonomy) that capture information about the knowledge domain and were never expressed explicitly. Now, as the number of associations of this kind can be very big, we employ CTCA for encoding and representing them *compactly*.

Valid		**Invalid**	
Earth, Sports	Greece, Sports	Crete, WinterSp.	Cefalonia, WinterSp.
Finland, Sports	Olympus, Sports	Rethimno, WinterSp.	Heraklio, WinterSp.
Crete, Sports	Cefalonia, Sports	Olympus, SeaSki	Olympus, WindSurf.
Rethimno, Sports	Heraklio, Sports	Crete, SnowBoard	Cefalonia, SnowBoard
Earth, SeaSports	Greece, SeaSports	Rethimno, SnowBoard	Heraklio, SnowBoard
Finland, SeaSports	Crete, SeaSports	Crete, SnowSki	Cefalonia, SnowSki
Cefalonia, SeaSports	Rethimno, SeaSports	Rethimno, SnowSki	Heraklio, SnowSki
Heraklio, SeaSports	Earth, WinterSp.	Finland, Cas.	Olympus, Cas.
Greece, WinterSp.	Finland, WinterSp.	Crete, Cas.	Heraklio, Cas.
Olympus, WinterSp.	Earth, SeaSki	Rethimno, Cas.	WinterSp., Cas.
Greece, SeaSki	Finland, SeaSki	SnowBoard, Cas.	SnowSki, Cas.
Crete, SeaSki	Cefalonia, SeaSki	Olympus, SeaSports	Crete, WinterSp., Cas.
Rethimno, SeaSki	Heraklio, SeaSki	Cefalonia, WinterSp., Cas.	Rethimno, WinterSp., Cas.
Earth, WindSurf.	Greece, WindSurf.	Heraklio, WinterSp., Cas.	Olympus, SeaSki, Cas.
Finland, WindSurf.	Crete, WindSurf.	Olympus, WindSurf., Cas.	Crete, SnowBoard, Cas.
Cefalonia, WindSurf.	Rethimno, WindSurf.	Cefalonia, SnowBoard, Cas.	Rethimno, SnowBoard, Cas.
Heraklio, WindSurf.	Earth, SnowBoard	Heraklio, SnowBoard, Cas.	Crete, SnowSki, Cas.
Greece, SnowBoard	Finland, SnowBoard	Cefalonia, SnowSki, Cas.	Rethimno, SnowSki, Cas.
Olympus, SnowBoard	Earth, SnowSki	Heraklio, SnowSki, Cas.	Olympus, Sports, Cas.
Greece, SnowSki	Finland, SnowSki	Crete, Sports, Cas.	Rethimno, Sports, Cas.
Olympus, SnowSki	Earth, Sports, Cas.	Heraklio, Sports, Cas.	Crete, SeaSports, Cas.
Greece, Sports, Cas.	Cefalonia, Sports, Cas.	Rethimno, SeaSports, Cas.	Heraklio, SeaSports, Cas.
Sports, Cas.	SeaSports, Cas.	Olympus, WinterSp., Cas.	Crete, SeaSki, Cas.
SeaSki, Cas.	Windsurf., Cas.	Rethimno, SeaSki, Cas.	Heraklio, SeaSki, Cas.
Earth, Cas.	Greece, Cas.	Crete, WindSurf., Cas.	Rethimno, WindSurf., Cas.
Cefalonia, Cas.	Earth, SeaSports, Cas.	Heraklio, WindSurf., Cas.	Olympus, SnowBoard, Cas.
Greece, SeaSports, Cas.	Earth, SeaSki, Cas.	Olympus, SnowSki, Cas.	Finland, Sports, Cas.
Greece, SeaSki, Cas.	Cefalonia, SeaSki, Cas.	Finland, SeaSports, Cas.	Finland, WinterSp., Cas.
Earth, WindSurf., Cas.	Greece, WindSurf., Cas.	Finland, SeaSki, Cas.	Finland, WindSurf., Cas.
Cefalonia, WindSurf., Cas.	Cefalonia, SeaSports, Cas.	Finland, SnowSki, Cas.	Finland, SnowBoard, Cas.
		Earth, WinterSp., Cas.	Greece, WinterSp., Cas.
		Earth, SnowBoard, Cas.	Greece, SnowBoard, Cas.
		Earth, SnowSki, Cas.	Greece, SnowSki, Cas.
		Olympus, SeaSports, Cas.	

Table 1: The valid and invalid compound terms of the example of Figure 1

This paper generalizes and extends the work first sketched in [24]. In particular in [24], we introduced the problem of expression mining and described some basic and unoptimized algorithms for solving it. Here, (i) we provide a thorough presentation of the problem of expression mining, including more expression mining subcases, and novel applications, (ii) we describe analytically the computational complexity of the unoptimized algorithms, presented in [24], and (iii) we introduce several optimization techniques along with their computational complexity, showing that they achieve improved performance.

The rest of this paper is organized as follows: Section 2 describes some indicative applications of expression mining. Section 3 describes the required background. Section 4 formulates the general problem of expression mining and identifies several specializations of it. Section 5 describes straightforward methods for expression mining, while Section 6 describes a method for finding the *shortest*, i.e. the most compact and efficient expression. Then, Section 7 elaborates expression mining in the most general form of the problem. Section 8 discusses the computa-

tional complexity of the algorithms for all of the above cases and Section 9 describes several optimizations. Finally, Section 10 summarizes and concludes the paper. All proofs are given in Appendix A. A table of symbols is given in Appendix B.

2 Applications of Expression Mining

Faceted classification was suggested quite long ago by Ranganathan in the 1920s [21]. Roughly, expression mining can be useful in every application that involves faceted classification, specifically in every application that involves materialized faceted taxonomies. The adoption of faceted taxonomies is beneficial not only for designing big Web Catalogs or Web Sites [19], but also for Libraries [15], Software Repositories [17, 18], and for other application domains, like biology[1]. Current interest in faceted taxonomies is also indicated by several recent or ongoing projects (like FATKS[2], FACET[3], FLAMENGO[4], and the emergence of XFML [1](Core-eXchangeable Faceted Metadata Language), a markup language for applying the faceted classification paradigm on the Web (for more about faceted classification see [21, 8, 32, 14, 19]).

Below we discuss in brief a number of indicative applications of expression mining.

Reusing the Knowledge of Materialized Faceted Taxonomies

Expression mining can be exploited for *encoding compactly* the set of valid compound terms of a materialized faceted taxonomy. The mined expressions allow exchanging and reusing this kind of knowledge in a very flexible and compact manner. One important remark is that the closed-world assumptions of the algebraic operations of CTCA (described in [25, 26]) lead to a remarkably high "compression ratio". It is also worthy to mention here that the recently emerged XFML+CAMEL (*C*ompound term composition *A*lgebraically-*M*otivated *E*xpression *L*anguage) [2] allows publishing and exchanging CTCA expressions using an XML format. As an example of expression mining, recall the scenario with the medical file that was mentioned in the introductory section.

Another application area is *symbolic data analysis* (SDA) [3, 6]. SDA has been introduced in order to solve the problem of the analysis of data that are given on an aggregated form, i.e. where quantitative variables are given by intervals and where categorical variables are given by histograms. This kind of data are generated when we summarize huge sets of data. Inescapably, even a symbolic data table could become very large in size, making its management problematic in terms of both storage space and computational time. As showed in [23] a symbolic data table can be seen as a materialized faceted taxonomy. Consequently, the algorithms that are presented in this paper can be used in order to compress a symbolic data table.

Reorganizing Single-Hierarchical Taxonomies to Faceted Taxonomies

Expression mining can also be exploited for *reorganizing* the big fat single-hierarchical taxonomies found on the Web (like Yahoo! or ODP) and in Libraries (like the majority of the thesauri [13]), so as to give them a clear, compact, and scalable faceted structure but *without missing* the knowledge that is hardwired in their structure and in their pre-coordinated vocabularies. Such a reorganization would certainly facilitate their management, extension, and reuse. Furthermore, it would allow the dynamic derivation of "complete" and meaningful navigational trees for this kind of sources (as described in detail in [27]), which unlike the existing naviga-

[1]The *Gene Ontology* (http://www.geneontology.org/) is a faceted taxonomy for indexing gene products.
[2]http://www.ucl.ac.uk/fatks/database.htm
[3]http://www.glam.ac.uk/soc/research/hypermedia/facet_proj/index.php
[4]http://bailando.sims.berkeley.edu/flamenco.html

tional trees of single-hierarchical taxonomies, do not present the problem of missing terms or missing relationships (for more about this problem see [4]). For example, for reusing the taxonomy of the Google directory, we now have to copy its entire taxonomy which currently consists of more than 450.000 terms and whose RDF representation[5] is a compressed file of 46 MBytes! According to our approach, we only have to *break* their pre-coordinated terms and then *partition* the resulting set of elemental terms to a set of facets, using languages like the one proposed in [22] (we shall not elaborate on this issue in this paper), and then use the algorithms presented in this paper for expression mining. Apart from smaller storage space requirements, the resulted faceted taxonomy can be modified/customized in a more flexible and efficient manner. Furthermore, a semantically clear, faceted structure can aid the manual or automatic construction of the inter-taxonomy mappings ([29]), which are needed in order to build mediators or peer-to-peer systems over this kind of sources [31, 30]. Figure 2 illustrates graphically this scenario.

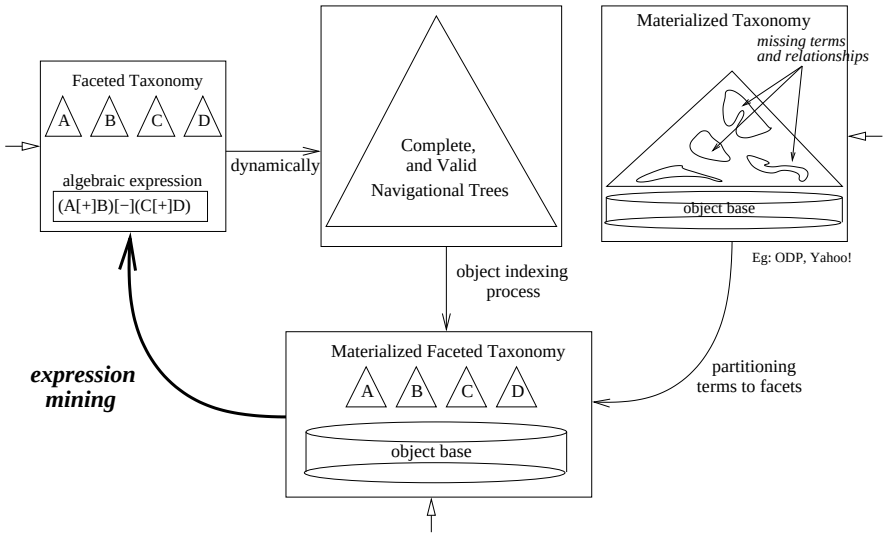

Figure 2: Some application scenarios of expression mining

Optimizing Query Answering

Suppose a materialized faceted taxonomy that stores a very large number of objects and users that pose conjunctive queries (i.e. compound terms) in order to find the objects of interest. The availability of an expression e that specifies the compound terms with non-empty interpretation would allow realizing efficiently (in polynomial time with respect to $|\mathcal{T}|$) whether a query has an empty or a non-empty answer. In this way, we can avoid performing wasteful computations for the queries whose answer will be eventually empty.

Language Engineering

Language Engineering is an area focusing on applications such as building natural language interfaces. This area employs ontologies, often called *lexical ontologies* or word-based thesauri, like WordNet [5] and Roget's thesaurus. They consist of terms and relationships between terms, where terms denote lexical entries (words), while relations are intended as lexical or semantic relations between terms. For instance in WordNet [5], terms are grouped into equivalence classes, called *synsets*. Each synset is assigned to a lexical category i.e. *noun*, *verb*, *adverb*, *adjective*, and synsets are linked by *hypernymy/hyponymy* and *antonymy* relations. The former is the subsumption relation, while the latter links together opposite or mutually inverse terms such

[5]Available at http://rdf.dmoz.org/

as tall/short, child/parent. Clearly, by ignoring the antonymy relation, the rest can be seen as a faceted taxonomy with the following facets: *noun*, *verb*, *adverb*, and *adjective*. This faceted taxonomy plus any collection of indexed subphrases from a corpus of documents is actually a materialized faceted taxonomy that contains information about the concurrence of words (e.g. which adjectives are applied to which nouns). This kind of knowledge might be quite useful as it can be exploited in several applications, e.g. for building a recommender system in a word processor (e.g. in MS Word), an automatic cross-language translator, or an information retrieval system (e.g. see [9]). Notice that in this scenario, the number of the valid combinations of words would be so big, that storing them explicitly would be quite problematic. Mining and storing only the CTCA expression that describes the set of all valid combinations, can definitely alleviate this problem. There might be also several applications related to the Generative Lexicon [20].

Sequences

Suppose that we are interested in sequences of k distinct steps where each step can be one of a finite set of actions. We can model all possible actions that can be enacted at one step as one taxonomy. In this way, we can model all possible sequences of k steps as a faceted taxonomy of k facets (facet i contains the actions that can be applied at the i-th step). Now assume that we have at our disposal a log file that stores a set of sequences that occurred in the past. These sequences may be those that have yielded a positive outcome. We might be interested in storing compactly these sequences, in order to build a recommending (or error avoiding) service. Expression mining can be applied in order to represent this set of sequences compactly. The T9 technique[6] (Text Input for Keypad Devices) that allows us to write quickly SMS messages from our mobile phones is a special case of this scenario.

3 Background

For self-containment, in the following two subsections, we briefly recall taxonomies, faceted taxonomies, compound taxonomies, and the Compound Term Composition Algebra. For more information and examples please refer to [27, 25, 26]. Subsequently, in subsection 3.3, we define materialized faceted taxonomies.

3.1 Taxonomies, Faceted Taxonomies, and Compound Taxonomies

A *taxonomy* is a pair $(\mathcal{T}, \leq)$, where $\mathcal{T}$ is a *terminology* and $\leq$ is a reflexive and transitive relation over $\mathcal{T}$, called *subsumption*. A *compound term* over $\mathcal{T}$ is any subset of $\mathcal{T}$. For example, the following sets of terms are compound terms over the taxonomy *Sports* of Figure 1: $s_1 = \{SeaSki, Windsurfing\}$, $s_2 = \{SeaSports, WinterSports\}$, $s_3 = \{Sports\}$, and $s_4 = \emptyset$. We denote by $P(\mathcal{T})$ the set of all compound terms over $\mathcal{T}$ (i.e. the powerset of $\mathcal{T}$). A *compound terminology* S over $\mathcal{T}$ is any set of compound terms that contains the compound term $\emptyset$. The set of all compound terms over $\mathcal{T}$ can be ordered using an ordering relation that is derived from $\leq$. Specifically, the *compound ordering* over $\mathcal{T}$ is defined as follows: if s, s' are compound terms over $\mathcal{T}$, then $s \preceq s'$ iff $\forall t' \in s' \quad \exists t \in s$ such that $t \leq t'$. That is, $s \preceq s'$ iff s contains a narrower term for every term of s'. In addition, s may contain terms not present in s'. Roughly, $s \preceq s'$ means that s carries more specific indexing information than s'. We say that two compound terms s, s' are *equivalent* iff $s \preceq s'$ and $s' \preceq s$. Intuitively, equivalent compound terms carry the same indexing information, and in our context are considered the same. A *compound taxonomy* over $\mathcal{T}$ is a pair $(S, \preceq)$, where S is a compound terminology over

[6] http://www.tegic.com/history.html, http://www.t9.com/

$\mathcal{T}$, and $\preceq$ is the compound ordering over $\mathcal{T}$ restricted to S. Clearly, $(P(\mathcal{T}), \preceq)$ is a compound taxonomy over $\mathcal{T}$. The broader and the narrower compound terms of a compound term s are defined as follows: $\mathrm{Br}(s) = \{s' \in P(\mathcal{T}) \mid s \preceq s'\}$ and $\mathrm{Nr}(s) = \{s' \in P(\mathcal{T}) \mid s' \preceq s\}$. The broader and the narrower compound terms of a compound terminology S are defined as follows: $Br(S) = \cup\{\mathrm{Br}(s) \mid s \in S\}$ and $Nr(S) = \cup\{\mathrm{Nr}(s) \mid s \in S\}$.

Let $\{F_1, ..., F_k\}$ be a finite set of taxonomies, where $F_i = (\mathcal{T}_i, \leq_i)$, and assume that the terminologies $\mathcal{T}_1, ..., \mathcal{T}_k$ are pairwise disjoint. Then, the pair $\mathcal{F} = (\mathcal{T}, \leq)$, where $\mathcal{T} = \cup_{i=1}^{k} \mathcal{T}_i$ and $\leq\ = \cup_{i=1}^{k} \leq_i$, is a taxonomy, which we shall call the *faceted taxonomy generated* by $\{F_1, ..., F_k\}$. We call the taxonomies $F_1, ..., F_k$ the *facets* of $\mathcal{F}$. Clearly, all definitions introduced so far apply also to faceted taxonomies. In particular, compound terms can be derived from a faceted taxonomy. For example, the set $S = \{\{Greece\}\ \{Sports\}, \{SeaSports\}, \{Greece, Sports\}, \{Greece, SeaSports\}, \emptyset\}$, is a compound terminology over the terminology $\mathcal{T}$ of the faceted taxonomy shown in Figure 1. The set S together with the compound ordering of $\mathcal{T}$ (restricted to S) is a compound taxonomy over $\mathcal{T}$. For reasons of brevity, hereafter we shall omit the term $\emptyset$ from the compound terminologies of our examples and figures.

3.2 The Compound Term Composition Algebra

Here we present in brief the *Compound Term Composition Algebra* (CTCA), an algebra for specifying the valid compound terms of a faceted taxonomy (for further details see [27, 25, 26]).

Let $\mathcal{F} = (\mathcal{T}, \leq)$ be a faceted taxonomy generated by a set of facets $\{F_1, ..., F_k\}$, where $F_i = (\mathcal{T}_i, \leq_i)$. The *basic compound terminology* of a terminology $\mathcal{T}_i$ is defined as follows: $T_i = \{\{t\} \mid t \in \mathcal{T}_i\} \cup \{\emptyset\}$. Note that each basic compound terminology is a compound terminology over $\mathcal{T}$. The basic compound terminologies $\{T_1, ..., T_k\}$ are the initial operands of the algebraic operations of CTCA.

The algebra includes four operations which allow combining terms from different facets, but also terms from the same facet. Two auxiliary *product* operations, one n-ary ($\oplus$) and one unary ($\overset{*}{\oplus}$), are defined to generate *all* combinations of terms from different facets and from one facet, respectively. Since not all term combinations are valid, more general operations are defined that include *positive* or *negative* modifiers, which are sets of known *valid* or known *invalid* compound terms. The unmodified product and self-product operations turn out to be special cases with the modifiers at certain extreme values. Specifically, the four basic operations of the algebra are: *plus-product* ($\oplus_P$), *minus-product* ($\ominus_N$), *plus-self-product* ($\overset{*}{\oplus}_P$), and *minus-self-product* ($\overset{*}{\ominus}_N$), where P denotes a set of valid compound terms and N denotes a set of invalid compound terms. The definition of each operation is given in Table 2.

Operation	e	S_e	arity
product	$S_1 \oplus ... \oplus S_n$	$\{\ s_1 \cup ... \cup s_n \mid s_i \in S_i\}$	n-ary
plus-product	$\oplus_P(S_1, ...S_n)$	$S_1 \cup ... \cup S_n\ \cup\ Br(P)$	n-ary
minus-product	$\ominus_N(S_1, ...S_n)$	$S_1 \oplus ... \oplus S_n - Nr(N)$	n-ary
self-product	$\overset{*}{\oplus}(T_i)$	$P(\mathcal{T}_i)$	unary
plus-self-product	$\overset{*}{\oplus}_P(T_i)$	$T_i \cup Br(P)$	unary
minus-self-product	$\overset{*}{\ominus}_N(T_i)$	$\overset{*}{\oplus}(T_i) - Nr(N)$	unary

Table 2: The operations of the Compound Term Composition Algebra

For example, consider the compound terminologies $S = \{\{Greece\}, \{Islands\}\}$ and $S' =$

$\{\{Sports\}, \{SeaSports\}\}$. The compound taxonomy corresponding to $S \oplus S'$ is shown in Figure 3, and consists of 9 compound terms. Recall that for reasons of brevity, we omit the term $\emptyset$ from the compound terminologies of our examples ($\emptyset$ is an element of S, S' and $S \oplus S'$).

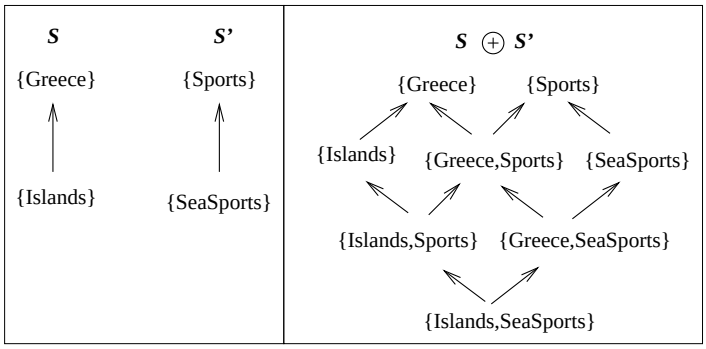

Figure 3: An example of a product $\oplus$ operation

Now consider the compound terminologies S and S' shown in the left part of Figure 4, and suppose that we want to define a compound terminology that does not contain the compound terms $\{Islands, WinterSports\}$ and $\{Islands, SnowSki\}$, because they are invalid. For this purpose instead of using a *product*, we have to use either a *plus-product* or a *minus-product* operation. Specifically, we can use a plus-product, $\oplus_P(S, S')$, where $P = \{\{Islands, Seasports\}, \{Greece, SnowSki\}\}$. The compound taxonomy defined by this operation is shown in the right part of Figure 4. In this figure, we enclose in squares the elements of P. We see that the compound terminology $\oplus_P(S, S')$ contains the compound term $s = \{Greece, Sports\}$, as $s \in Br(\{Islands, SeaSports\})$. However, it does not contain the compound terms $\{Islands, WinterSports\}$ and $\{Islands, SnowSki\}$, as they do not belong to $S \cup S' \cup Br(P)$ (see the definition of plus-product in Table 2). Alternatively, we can obtain the compound taxonomy shown at the right part of Figure 4 by using a *minus-product* operation, i.e. $\ominus_N(S, S')$, with $N = \{\{Islands, WinterSports\}\}$. The result does not contain the compound terms $\{Islands, WinterSports\}$ and $\{Islands, SnowSki\}$, as they are elements of $Nr(N)$ (see the definition of minus-product in Table 2).

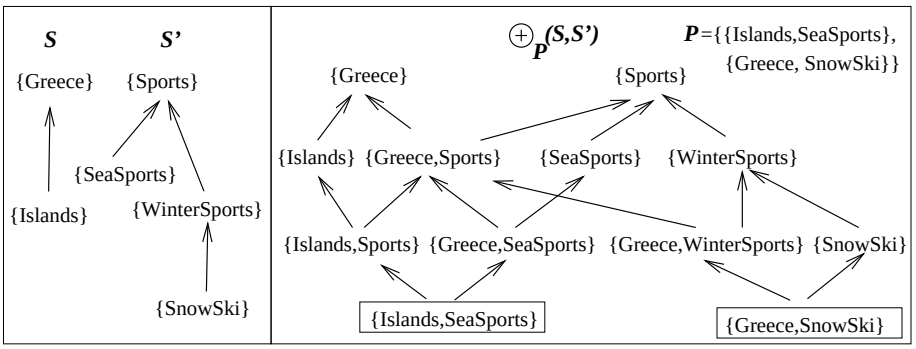

Figure 4: An example of a *plus-product*, $\oplus_P$, operation

Def 3.1 An expression e over $\mathcal{F}$ is defined according to the following grammar ($i = 1, ..., k$):

$$e ::= \oplus_P(e, ..., e) \mid \ominus_N (e, ..., e) \mid \overset{*}{\oplus}_P T_i \mid \overset{*}{\ominus}_N T_i \mid T_i,$$

where the parameters P and N denote sets of valid and invalid compound terms, respectively.

The outcome of the evaluation of an expression e is denoted by S_e, and is called the *compound terminology* of e. In addition, $(S_e, \preceq)$ is called the *compound taxonomy* of e.

Let $\mathcal{T}_e$ be the union of the terminologies of the facets appearing in an expression e. The expression e actually partitions the set $P(\mathcal{T}_e)$ into two sets:
(a) the set of valid compound terms S_e, and
(b) the set of invalid compound terms $P(\mathcal{T}_e) - S_e$.

To proceed we need to distinguish what we shall call *genuine compound terms*. Intuitively, a genuine compound term combines non-empty compound terms from *more than one* compound terminology. Specifically, the set of *genuine compound terms* over a set of compound terminologies $S_1, ..., S_n$ is defined as follows: $G_{S_1,...,S_n} = S_1 \oplus ... \oplus S_n - \cup_{i=1}^{n} S_i$. For example, if $S_1 = \{\{Greece\}, \{Islands\}\}$, $S_2 = \{\{Sports\}, \{WinterSports\}\}$, and $S_3 = \{\{Pensions\}, \{Hotels\}\}$ then: $\{Greece, WinterSports, Hotels\} \in G_{S_1,S_2,S_3}$, $\{WinterSports, Hotels\} \in G_{S_1,S_2,S_3}$, but $\{Hotels\} \notin G_{S_1,S_2,S_3}$. Additionally, the set of *genuine compound terms* over a basic compound terminology T_i, $i = 1, ..., k$, is defined as follows: $G_{T_i} = \overset{*}{\oplus}(T_i) - T_i$. The sets of genuine compound terms are used to define a *well-formed* algebraic expression.

An expression e is *well-formed* iff:
(i) each basic compound terminology T_i appears at most once in e,
(ii) each parameter P that appears in e, is a subset of the associated set of genuine compound terms, e.g. if $e = \oplus_P(e_1, e_2)$ or $e = \overset{*}{\oplus}_P(T_i)$ then it should be $P \subseteq G_{S_{e_1},S_{e_2}}$ or $P \subseteq G_{T_i}$, respectively, and
(iii) each parameter N that appears in e, is a subset of the associated set of genuine compound terms, e.g. if $e = \ominus_N(e_1, e_2)$ or $e = \overset{*}{\ominus}_N(T_i)$ then it should be $N \subseteq G_{S_{e_1},S_{e_2}}$ or $N \subseteq G_{T_i}$, respectively.

In the rest of the paper, *we consider only well-formed expressions*. In [27], we presented the algorithm $IsValid(e, s)$ that checks the validity of a compound term s according to a well-formed expression e in $O(|\mathcal{T}|^2 * |s| * |\mathcal{P} \cup \mathcal{N}|)$ time, where $\mathcal{P}$ denotes the union of all P parameters and $\mathcal{N}$ denotes the union of all N parameters appearing in e. Additionally, in [26] we showed why we cannot use Description Logics [7] for expressing what we can with the Compound Term Composition Algebra. At last we should mention that a system that supports the design of faceted taxonomies and the interactive formulation of CTCA expressions has already been implemented by VTT and Helsinki University of Technology (HUT) under the name FASTAXON [28]. The system is currently under experimental evaluation.

3.3 Materialized Faceted Taxonomies

Let Obj denote the set of all objects of our domain, e.g. the set of all hotel Web pages. An *interpretation* of a set of terms $\mathcal{T}$ over Obj is any (total) function $I : \mathcal{T} \rightarrow P(Obj)$.

A *materialized faceted taxonomy* M is a pair $(\mathcal{F}, I)$, where $\mathcal{F} = (\mathcal{T}, \leq)$ is a faceted taxonomy, and I is an interpretation of $\mathcal{T}$. An example of a materialized faceted taxonomy is given in Figure 5 where the objects are denoted by natural numbers. This will be the running example of our paper.

Apart from browsing, we can also query a materialized faceted taxonomy. A simple query language is introduced next. A *query* over $\mathcal{T}$ is any string derived by the following grammar: $q ::= t \mid q \wedge q' \mid q \vee q' \mid q \wedge \neg q' \mid (q) \mid \epsilon$, where t is a term of $\mathcal{T}$. Now let $Q_{\mathcal{T}}$ denote the set of all queries over $\mathcal{T}$. Any interpretation I of $\mathcal{T}$ can be extended to an interpretation $\hat{I}$ of $Q_{\mathcal{T}}$ as follows: $\hat{I}(t) = I(t)$, $\hat{I}(q \wedge q') = \hat{I}(q) \cap \hat{I}(q')$, $\hat{I}(q \vee q') = \hat{I}(q) \cup \hat{I}(q')$, $\hat{I}(q \wedge \neg q') = \hat{I}(q) - \hat{I}(q')$. One can easily see that a compound term $\{t_1, ..., t_k\}$ actually corresponds to a conjunction $t_1 \wedge ... \wedge t_k$.

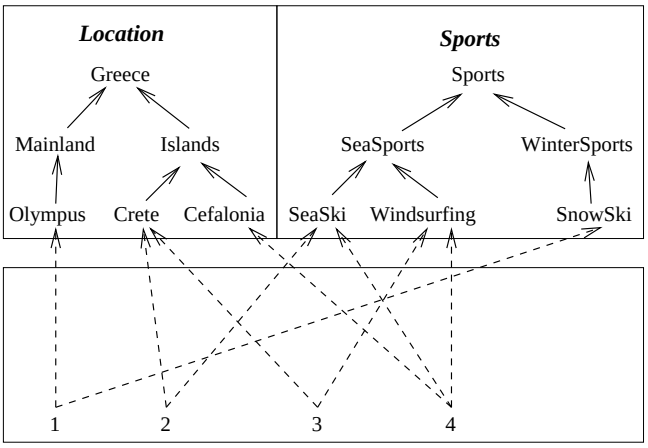

Figure 5: A materialized faceted taxonomy

However, in order for answers to make sense, the interpretation used for answering queries must respect the structure of the faceted taxonomy in the following intuitive sense: if $t \leq t'$ then $I(t) \subseteq I(t')$. The notion of model, introduced next, captures well-behaved interpretations. An interpretation I is a *model* of a taxonomy $(\mathcal{T}, \leq)$ if for all t, t' in $\mathcal{T}$, if $t \leq t'$ then $I(t) \subseteq I(t')$.

Given an interpretation I of $\mathcal{T}$, the model of $(\mathcal{T}, \leq)$ *generated* by I, denoted $\bar{I}$, is given by: $\bar{I}(t) = \cup\{I(t') \mid t' \leq t\}$. Now the answer of a query q is the set of objects $\hat{\bar{I}}(q)$. For instance, in our running example we have $\hat{\bar{I}}(Islands) = \{2, 3, 4\}$, $\hat{\bar{I}}(\{Crete, SeaSki\}) = \{2\}$, and $\hat{\bar{I}}(\{SeaSki, Windsurfing\}) = \{4\}$.

4 Problem Statement

The *set of valid compound terms* of a materialized faceted taxonomy $M = (\mathcal{F}, I)$ is defined as:

$$V(M) = \{s \in P(\mathcal{T}) \mid \hat{\bar{I}}(s) \neq \emptyset\}$$

where $\bar{I}$ is the model of $(\mathcal{T}, \leq)$ generated by I.[7]

As in the majority of applications, we are interesting in the valid compound terms that contain *at most* one term from each facet, we shall use $V_\circ(M)$ to denote this subset of $V(M)$. One can easily see that it holds:

$$V_\circ(M) = V(M) \cap (T_1 \oplus ... \oplus T_k)$$

For example, the following table indicates (by an $\surd$) the elements of $V_\circ(M)$ of the materialized faceted taxonomy shown in Figure 5.

	Greece	Mainland	Olympus	Islands	Crete	Cefalonia
Sports	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$
SeaSports	$\surd$			$\surd$	$\surd$	$\surd$
SeaSki	$\surd$			$\surd$	$\surd$	$\surd$
Windsurfing	$\surd$			$\surd$	$\surd$	$\surd$
WinterSports	$\surd$	$\surd$	$\surd$			
SnowSki	$\surd$	$\surd$	$\surd$			

Our initial problem of *expression mining* is formulated as follows:

[7] As all single terms of a faceted taxonomy are meaningful, we assume that $V(M)$ contains all singleton compound terms.

Problem 1: *Given a materialized faceted taxonomy $M = (\mathcal{F}, I)$, find an expression e over $\mathcal{F}$ such that $S_e = V_\circ(M)$.*

One can easily see, that as we work with $V_\circ(M)$, the expression that we are looking for does not contain any self-product operation.

Let us define the *size* of an expression e as follows: $size(e) = |\mathcal{P} \cup \mathcal{N}|$, where $\mathcal{P}$ denotes the union of all P parameters of e, and $\mathcal{N}$ denotes the union of all N parameters of e. Among the expressions e that satisfy $S_e = V_\circ(M)$, we are more interested in finding the *shortest* expression. This is because, in addition to smaller space requirements, the time needed for checking compound term validity according to the mined expression e is reduced[8]. Reducing the time needed for checking compound term validity, improves the performance of several on-line tasks associated with knowledge reuse. Indeed, as it was shown in [27], the algorithm $IsValid(e, s)$ is called during the dynamic construction of the navigation tree that guides the indexer and the final user through his/her (valid) compound term selection. Though, shortest expression mining is a costly operation, it is not a routine task. Therefore, we consider that reducing the size of the mined expression is more important than reducing the time needed for its extraction. In particular, we are interested in the following problem:

Problem 2: *Given a materialized faceted taxonomy $M = (\mathcal{F}, I)$, find the <u>shortest</u> expression e over $\mathcal{F}$ such that $S_e = V_\circ(M)$.*

One important remark is that solving Problem 2 allow us to solve also the following:

Problem 3: *Given an expression e' that does not contain any self-product operation, find the shortest expression e such that $S_e = S_{e'}$.*

One can easily see that the same algorithms can be used for solving both Problem 2 and Problem 3. The only difference is that, in the second problem we have to consider that $V_\circ(M)$ is the set $S_{e'}$. Note that this kind of "optimization" could be very useful even during the design process, i.e. a designer can use several times the above "optimizer" during the process of formulating an algebraic expression.

Let us now consider the general case, in which we are also interested in the compound terms that contain more than one term from each facet. Here, expression mining is formulated as:

Problem 4: *Given a materialized faceted taxonomy $M = (\mathcal{F}, I)$, find the <u>shortest</u> expression e over $\mathcal{F}$ such that $S_e = V(M)$.*

One can easily see that here the expression that we are looking for may contain one or more self-product operations.

Furthermore, we can generalize Problem 3 as follows:

Problem 5: *Given an expression e', find the shortest expression e such that $S_e = S_{e'}$.*

In Section 5 we tackle Problem 1, in Section 6 we tackle Problems 2 and 3, and finally, in Section 7 we tackle Problems 4 and 5.

[8]Recall that the time complexity of Alg. $IsValid(e, s)$ [27] is proportional to $size(e) = |\mathcal{P} \cup \mathcal{N}|$.

5 Mining an Expression

One straightforward method to solve Problem 1 is to find an expression e with only one plus-product operation over the basic compound terminologies $T_1, ..., T_k$, i.e. an expression of the form: $e = \oplus_P(T_1, ..., T_k)$.

We can compute the parameter P of this operation in two steps:

1. $P' := V(M) \cap G_{T_1,...,T_k}$
2. $P := minimal(P')$

The first step computes all valid compound terms that (a) contain at most one term from each facet, and (b) do not belong to the basic compound terminologies (i.e., are not singletons). Algorithmically, we can do this as follows: For all $s \in G_{T_1,...,T_k}$, if $\hat{\bar{I}}(s) \neq \emptyset$, then we store s in P'. One can easily see that $S_{\oplus_{P'}(T_1,...,T_k)} = V_\circ(M)$. The second step is optional and aims at reducing the size of the mined expression. Specifically, it eliminates the redundant compound terms of the parameter P', i.e. those compound terms that are not minimal (w.r.t. $\preceq$). This is because, it holds: $\oplus_{P'}(T_1, ..., T_k) = \oplus_{minimal(P')}(T_1, ..., T_k)$. Thus, if P contains the minimal elements of P', then it is evident that again it holds $S_{\oplus_P(T_1,...,T_k)} = V_\circ(M)$. By applying the above two-step algorithm to our current example (of Figure 5) we get that: $P = \{\{Olympus, SnowSki\}, \{Crete, SeaSki\}, \{Crete, Windsurfing\}, \{Cefalonia, SeaSki\}, \{Cefalonia, Windsurfing\}\}$.

Analogously, we can find an expression e with only one minus-product operation over the basic compound terminologies $T_1, ..., T_k$, i.e. an expression of the form: $e = \ominus_N(T_1, ..., T_k)$.

We can compute the parameter N of this operation in two steps:

1. $N' := G_{T_1,...,T_k} - V(M)$
2. $N := maximal(N')$

The first step computes all invalid compound terms that contain at most one term from each facet. Algorithmically, we can do this as follows: For all $s \in G_{T_1,...,T_k}$, if $\hat{\bar{I}}(s) = \emptyset$ then we store s in N'. One can easily see that $S_{\ominus_{N'}(T_1,...,T_k)} = V_\circ(M)$. Again, the second step is optional and aims at reducing the size of the mined expression. Specifically, it eliminates the redundant compound terms, i.e. compound terms that are not maximal (w.r.t. $\preceq$). This is because, it holds: $\ominus_{N'}(T_1, ..., T_k) = \ominus_{maximal(N')}(T_1, ..., T_k)$. Thus, if N contains the maximal elements of N', then it is evident that again it holds $S_{\ominus_N(T_1,...,T_k)} = V_\circ(M)$. By applying the above two-step algorithm to our current example we get that: $N = \{\{Mainland, SeaSports\}, \{Islands, WinterSports\}\}$. Notice that here the size of the minus-product expression is smaller than the size of the plus-product expression (the parameter N contains only 2 compound terms, while P contains 5). We refer to the above techniques are *straightforward expression mining*.

6 Mining the Shortest Expression

Let us now turn our attention to Problem 2, i.e. on finding the *shortest* expression e over a given materialized faceted taxonomy $M = (\mathcal{F}, I)$ such that $S_e = V_\circ(M)$.

At first notice that since our running example has only two facets, the shortest expression is either a plus-product or a minus-product operation. However, in the general case where we have several facets, finding the shortest expression is more complicated because there are several forms that an expression can have.

Below we present the algorithm $FindShortestExpression(M)$ (Alg. 6.1) which takes as input a materialized faceted taxonomy $M = (\mathcal{F}, \leq)$, and returns the *shortest* expression e over $\mathcal{F}$ such that $S_e = V_\circ(M)$. It is an exhaustive algorithm, in the sense that it investigates all *forms* that an expression over $\mathcal{F}$ may have. We use the term *expression form* to refer to an algebraic expression whose P and N parameters are undefined (unspecified). Note that an expression form can be represented as a parse tree.

Specifically, algorithm $FindShortestExpression(M)$ calls the procedure $ParseTrees(\{F_1, ..., F_k\})$ (described in subsection 6.1) which takes as input the set of facet names $\{F_1, ..., F_k\}$, and returns all possible parse trees of the expressions over $\{F_1, ..., F_k\}$.

Now the procedure $SpecifyParams(e, M)$ (described in subsection 6.2) takes as input a parse tree e returned by $ParseTrees(\{F_1, ..., F_k\})$ and the materialized faceted taxonomy M, and *specifies* the parameters P and N of e such that $S_e = V_\circ(M)$.

The procedure $GetSize(e)$ takes as input an expression e and returns the *size* of e, i.e. $|\mathcal{P} \cup \mathcal{N}|$.[9]

Algorithm $FindShortestExpression(M)$ returns the shortest expression e such that $S_e = V_\circ(M)$. Therefore, $FindShortestExpression(M)$ returns the solution to Problem 2.

Algorithm 6.1 $FindShortestExpression(M)$
Input: A materialized faceted taxonomy M, with facets $\{F_1, ..., F_k\}$
Output: The shorthest expression e such that $S_e = V_\circ(M)$

```
(1)  minSize := MAXINT;   // MAXINT is the largest representable integer
(2)  shortestExpr := "";
(3)  For each e in ParseTrees({F1, ..., Fk}) do
(4)      e' := SpecifyParams(e, M);
(5)      size := GetSize(e');
(6)      If size < minSize then
(7)          minSize := size;
(8)          shortestExpr := e';
(9)      EndIf
(10) EndFor
(11) return (shortestExpr)
```

6.1 Deriving all Possible Parse Trees

The algorithm $ParseTrees(\{F_1, ..., F_k\})$ takes as input the set of facets names $\{F_1, ..., F_k\}$, and returns all possible parse trees of the expressions over $\{F_1, ..., F_k\}$. This algorithm in described in detail in [24]. For example, Figure 6 displays the parse trees returned by $ParseTrees(\{A\})$, $ParseTrees(\{A, B\})$, and $ParseTrees(\{A, B, C\})$. Note that, the *leaf* nodes of a parse tree are always facet names[10]. Additionally, the *internal* nodes of a parse tree are named "+" or "-", corresponding to a plus-product ($\oplus_P$) or a minus-product ($\ominus_N$) operation, respectively. As we consider only well-formed expressions, every facet name appears just once in a parse tree.

The algorithm $ParseTrees(\{F_1, ..., F_k\})$ [24] is a recursive algorithm that first calls $ParseTrees(\{F_1, ..., F_{k-1}\})$. Then, it extends each parse tree returned by $ParseTrees(\{F_1, ...,$

[9] We could also employ a more refined measure for the size of an expression, namely the exact number of terms that appear in the parameter sets, i.e. $size(e) = \sum_{s \in \mathcal{P} \cup \mathcal{N}} |s|$, where $|s|$ denotes the number of terms of the compound term s.

[10] Specifically, a terminal node with name F_i corresponds to the basic compound terminology T_i.

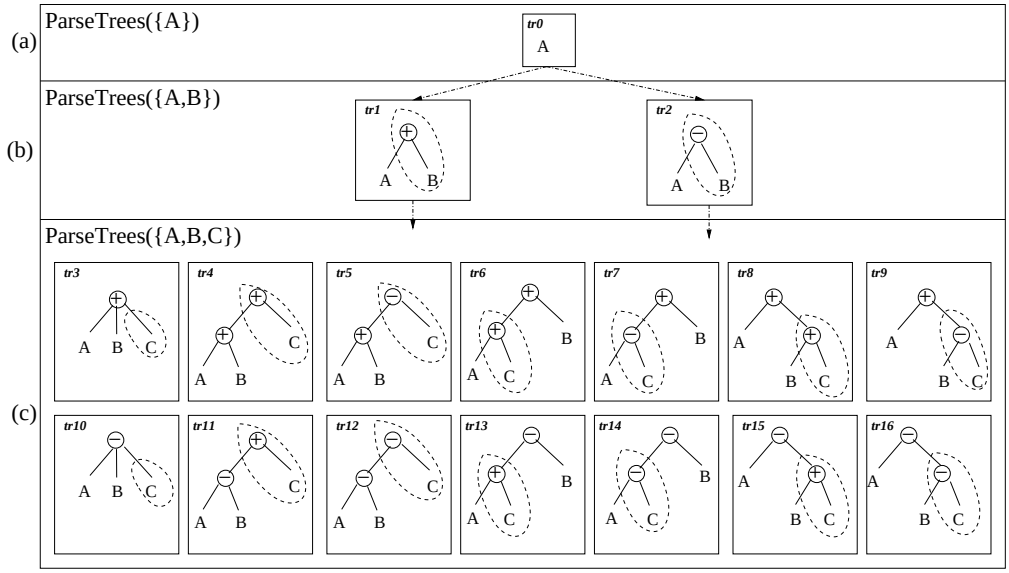

Figure 6: All possible parse trees for $\{A\}$, $\{A, B\}$, and $\{A, B, C\}$

$F_{k-1}\}$) with the new facet name F_k (in all possible ways). For example, the parse trees in the first row of Figure 6(c) are derived by extending the parse tree $tr1$ (returned by $ParseTrees(\{A, B\})$) with the facet name C. Dashed circles enclose the particular kind of extension. Similarly, the parse trees in the second row of Figure 6(c) are derived by extending the parse tree $tr2$ with the facet name C.

Let's now count the parse trees returned by $ParseTrees(\{F_1, ..., F_k\})$. The following proposition gives us an upper bound.

Prop. 1 Let PT_k be the number of parse trees returned by $ParseTrees(\{F_1, ..., F_k\})$. It holds that $PT_k \leq (k-1)! * 5^{k-1}$.

We want to note that this upper bound is overpessimistic, i.e. much bigger than the actual number of parse trees (e.g. $PT_4 = 162$, while $3!5^3 = 750$).

6.2 Specifying the Parameters

This section describes the algorithm $SpecifyParams(e, M)$ (Alg. 6.2), i.e. an algorithm that takes as input the parse tree of an expression e (with undefined P and N parameters) and a materialized faceted taxonomy M, and returns the same parse tree but now enriched with parameters P and N that satisfy the condition $S_e = V_\circ(M)$.

The algorithm works as follows. Suppose that the current node is an internal node that corresponds to a plus-product operation $\oplus_P(e_1, ..., e_n)$. For setting the parameter P of this operation we must first define the parameters of all subexpressions e_i, for all $i = 1, ..., n$. Therefore, the procedure $SpecifyParams(e_i, M)$ is called recursively, for all $i = 1, ..., n$. Subsequently, the statement $P' := G_{S_{e_1},...,S_{e_n}} \cap V_\circ(M)$ computes and stores in P' those elements of $V_\circ(M)$ that also belong to $G_{S_{e_1},...,S_{e_n}}$ (recall constraint (ii) of a well-formed expression). Finally, P is set equal to the minimal compound terms of P' (for the reasons described in Section 5).

Now suppose that the current node is an internal node that corresponds to a minus-product operation $\ominus_N(e_1, ..., e_n)$. Again, before defining N we have to define the parameters of all subexpressions e_i, for all $i = 1, ..., n$. So, the procedure $SpecifyParams(e_i, M)$ is called recursively,

Algorithm 6.2 $SpecifyParams(e, M)$
Input: The parse tree of an expression e, and a materialized faceted taxonomy M
Output: The parse tree of e enriched with P and N parameters, such that $S_e = V_\circ(M)$

(1) case(e) {
(2) $\oplus_P(e_1, ..., e_n)$: For $i := 1, ..., n$ do
(3) $e_i :=$ **SpecifyParams**(e_i, M) ;
(4) $P' := G_{S_{e_1},...,S_{e_n}} \cap V_\circ(M)$;
(5) $e.P :=$ **minimal**(P') ;
(6) return(e)
(7) $\ominus_N(e_1, ..., e_n)$: For $i := 1, ..., n$ do
(8) $e_i :=$ **SpecifyParams**(e_i, M) ;
(9) $N' := G_{S_{e_1},...,S_{e_n}} - V_\circ(M)$;
(10) $e.N :=$ **maximal**(N') ;
(11) return(e)
(12) T_i: return(e)
(13) }

for all $i = 1, ..., n$. Subsequently, the statement $N' := G_{S_{e_1},...,S_{e_n}} - V_\circ(M)$ computes and stores in N' those elements of $G_{S_{e_1},...,S_{e_n}}$ that are invalid, i.e. not in $V_\circ(M)$ (recall constraint (iii) of a well-formed expression). Finally, N is set equal to the maximal compound terms of N' (for the reasons described in Section 5).

6.3 An Example of Shortest Expression Mining

Let us now apply the above algorithms to a materialized faceted taxonomy $M = (\mathcal{F}, I)$, where $\mathcal{F}$ is the four-faceted taxonomy shown in Figure 7(a). The set of all compound terms that consist of at most one term from each facet are $(|A| + 1) * (|B| + 1) * (|C| + 1) * (|D| + 1) = 108$. Now let us suppose that the set of valid compound terms $V_\circ(M)$ consists of the 48 compound terms listed in Figure 7(b). For simplification, in that figure we do not show the interpretation I, but directly the set $V_\circ(M)$.

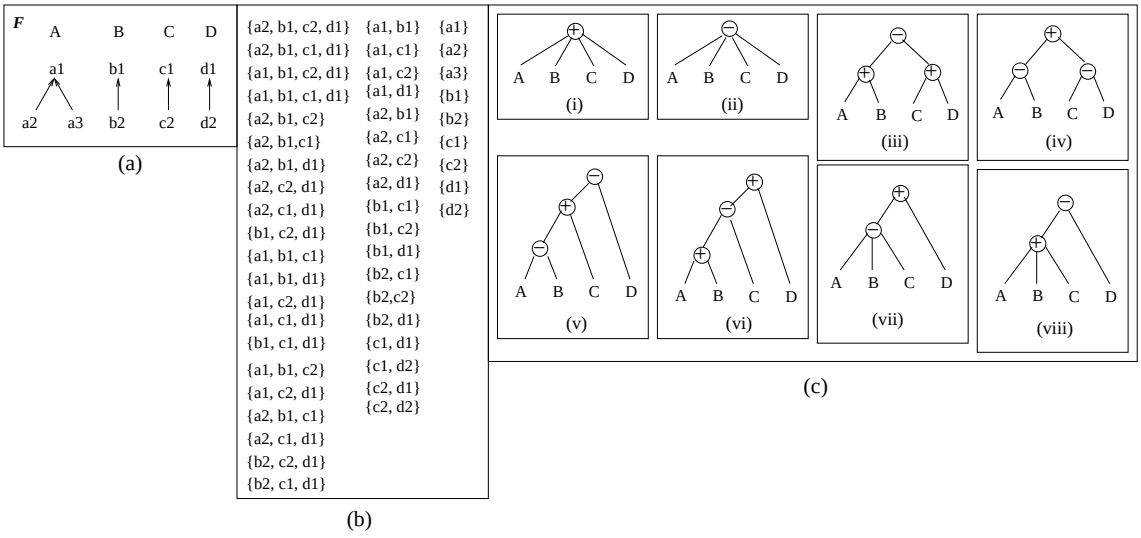

Figure 7: An example of shortest expression mining

The algorithm $FindShortestExpression(M)$ calls the procedure $ParseTrees(\{A, B, C, D\})$, to get the parse trees of all possible expressions over the facets $\{A, B, C, D\}$ (Figure 7(c) sketches

some indicative parse trees). Then, for each parse tree e in the output, it calls the procedure $SpecifyParams(e, M)$, which assigns particular values to the parameters P and N of e such that $S_e = V_\circ(M)$. The sizes of all derived expressions are compared to get the shortest expression, which is the following: $e = \oplus_P(A, \ominus_N(B, C, D))$ where $P = \{\{a2, b1, c2, d1\}\}$ and $N = \{\{b2, d2\}\}$. Notice that the size of $|\mathcal{P} \cup \mathcal{N}|$ is only 2! Even from this toy example it is evident that the expressions of the Compound Term Composition Algebra can indeed encode very *compactly* the knowledge of materialized taxonomies. As we have already mentioned, this is due to the closed world assumptions of the algebraic operations.

7 Shortest Expression Mining in the Most General Case

In this section we consider the general case, where we are interested in all valid compound terms, i.e. in the whole $V(M)$. Expression mining in this case requires employing self-product operations and extending the algorithms discussed in the previous section. This is because a compound term in $V(M)$ may contain more than one term from the same facet.

The extension of the algorithms is quite straightforward. The basic idea is to extend the parse trees returned by $ParseTrees(\{F_1, ..., F_k\})$ such that on top of each leaf node F_i, $i = 1, ..., k$, there is always a node that corresponds to either a *plus-self-product* ($\overset{*}{\oplus}_P(T_i)$) or a *minus-self-product* ($\overset{*}{\ominus}_N(T_i)$) operation. For example, Figure 8 shows the parse trees of all expressions (a) over a facet $\{A\}$ and (b) over two facets $\{A, B\}$. Notice that in the figure, we use $s\oplus$ and $s\ominus$ to indicate the nodes that correspond to a *plus-self-product* and *minus-self-product* operation, respectively.

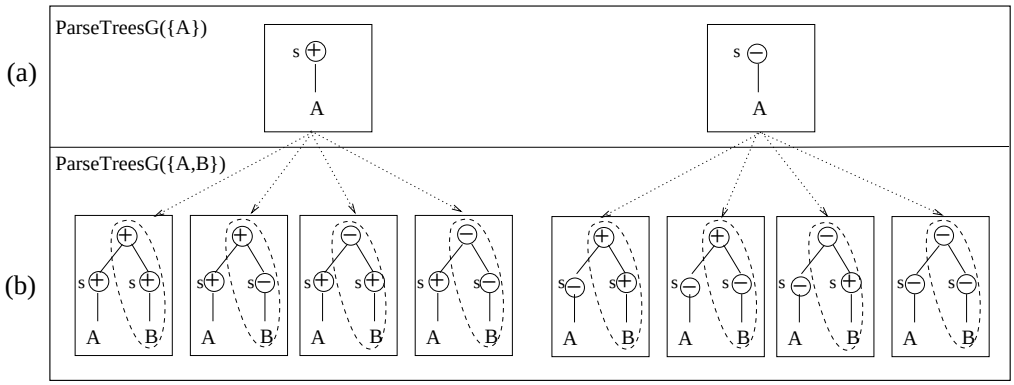

Figure 8: All possible parse trees for $\{A\}$ and $\{A, B\}$

Finally, algorithm $SpecifyParamsG$ (Alg. 7.1) is an extension of the algorithm $SpecifyParams$ (Alg. 6.2). It includes two extra cases, one for each self-product operation. The treatment of these cases is analogous to that of the product operations. Additionally, $V_\circ(M)$ has been replaced by $V(M)$.

8 The Computational Complexity of Expression Mining

In this section, we will measure the computational complexity of expression mining. At first we will measure the complexity of some basic decision problems, then of the algorithms for straightforward expression mining, and finally, of the algorithms for shortest expression mining.

Algorithm 7.1 *SpecifyParamsG*(e, M)
Input: The parse tree of an expression e, and a materialized faceted taxonomy M
Output: The parse tree of e enriched with P and N parameters, such that $S_e = V(M)$

(1) case(e) {
(2) $\oplus_P(e_1, ..., e_n)$: For $i := 1, ..., n$ do
(3) $e_i :=$ **SpecifyParamsG**(e_i, M) ;
(4) $P' := G_{S_{e_1},...,S_{e_n}} \cap V(M)$;
(5) $e.P :=$ **minimal**(P') ;
(6) return(e)
(7) $\ominus_N(e_1, ..., e_n)$: For $i := 1, ..., n$ do
(8) $e_i :=$ **SpecifyParamsG**(e_i, M) ;
(9) $N' := G_{S_{e_1},...,S_{e_n}} - V(M)$;
(10) $e.N :=$ **maximal**(N') ;
(11) return(e)
(12) $\boxed{\overset{*}{\oplus}_P\ (T_i)\text{:}}$ $P' := G_{T_i} \cap V(M)$;
(13) $e.P :=$ **minimal**(P');
(14) return(e)
(15) $\boxed{\overset{*}{\ominus}_N\ (T_i)\ \text{:}}$ $N' := G_{T_i} - V(M)$;
(16) $e.N :=$ **maximal**(N');
(17) return(e)
(18) }

At first we introduce some auxiliary definitions and some preliminary results. We define the *source graph* $\Gamma = (N, E)$ of a materialized faceted taxonomy $M = (\mathcal{F}, I)$ as follows:

$$\begin{aligned} N &= \mathcal{T} \cup Obj \\ E &= \{\ (t, t') \mid t \leq t'\} \bigcup \{\ (o, t) \mid o \in I(t)\} \end{aligned}$$

Roughly, the source graph is obtained by taking the union of the faceted taxonomy $\mathcal{F}$ and I^{-1}, where I^{-1} is a function from Obj to $P(\mathcal{T})$ such that: $t \in I^{-1}(o)$ if and only if $o \in I(t)$. For example, the source graph of the materialized faceted taxonomy shown in Figure 5, coincides with the illustrated directed graph in the figure. Clearly, a materialized faceted taxonomy M takes $\mathcal{O}((|\mathcal{T}|+|Obj|)^2)$ storage space[11].

Let us now turn our attention to compound terms and expression mining. One can easily see that the maximum number of valid compound terms of a materialized faceted taxonomy M, i.e. the quantity $|V(M)|$, can be $2^{|\mathcal{T}|}$. Now the maximum number of compound terms with at most one term from each facet, i.e. $|V_\circ(M)|$, can be equal to $|T_1 \oplus ... \oplus T_k|$, i.e. $(|\mathcal{T}_1|+1)*...*(|\mathcal{T}_k|+1)$. Let us now investigate how big $(|\mathcal{T}_1| + 1) * ... * (|\mathcal{T}_k| + 1)$ can be in terms of $|\mathcal{T}|$ and k. This requires solving the following maximization problem:

$$max_{x_1,...,x_k}\{\prod_{i=1}^{k}(1 + x_i) \mid \sum_{i=1}^{k} x_i = C\}\ \text{, where}\ \ C = |\mathcal{T}|$$

We can solve this problem using the method of *Lagrange Multipliers*, i.e. by assuming that we want to maximize the function $f(x) = \prod_{i=1}^{k}(1 + x_i)$ subject to the constraint $g(x) = \sum_{i=1}^{k} x_i = C$, where $x = (x_1, ..., x_k)$ and $C = |\mathcal{T}|$. This requires solving a system of $k + 1$ equations.

[11] Of course, by Obj we mean the objects of M, and not of the whole domain.

Specifically, we have the following k equations:

$$\frac{\partial}{\partial x_i}(f(x) + \lambda g(x)) = 0, \;\; 1 \leq i \leq k, \;\; \lambda \in \Re$$

and the equation $\sum_{i=1}^{k} x_i = C$. From the first k equations, it follows that $x_1 = x_2 = ... = x_k$. Then, by considering the last equation it follows that $x_i = \frac{C}{k} = \frac{|\mathcal{T}|}{k}$. This means that $G_{T_1,...,T_k}$ has at most $(1 + \frac{|\mathcal{T}|}{k})^k$ elements. Note that for $k = 1$ we get $1 + |\mathcal{T}|$, and for $k = |\mathcal{T}|$ we get $2^{|\mathcal{T}|}$. Thus, $|T_1 \oplus ... \oplus T_k|$ is in the magnitude of $(\frac{|\mathcal{T}|}{k})^k$.

8.1 Complexity of some Basic Decision Problems

Here, we will elaborate on the following decision problems, where o is an object, t a term, and s a compound term: (a) $o \overset{?}{\in} \bar{I}(t)$, (b) $o \overset{?}{\in} \hat{\bar{I}}(s)$, (c) $\overset{?}{\exists}$ o such that $o \in \hat{\bar{I}}(s)$, and (d) $s \overset{?}{\preceq} s'$.

Hereafter, we shall add the subscript "$_\circ$" to the id of a task, in order to denote the same task but with the additional assumption that we are interested only in $V_\circ(M)$, i.e. in the compound terms that contain at most one term from each facet. All complexity results of this subsection are summarized in Table 3.

Decision problem (a) ($o \overset{?}{\in} \bar{I}(t)$) reduces to the problem of REACHABILITY in graphs. Specifically, $o \in \bar{I}(t)$ if and only if the node that corresponds to term t (denoted by n_t) is reachable from the node that corresponds to object o (denoted by n_o). Thus, we can write $o \in \bar{I}(t)$ iff $n_o \rightarrow^* n_t$. As a consequence, the time complexity of this problem is $\mathcal{O}(|E|)$, where $|E|$ is the number of processed edges (for more see [16]). Note that the objects (i.e., the elements of Obj) are not connected with other objects (but only with terms), thus at most $|\mathcal{T}|^2 + |\mathcal{T}|$ edges have to processed. Thus, the time complexity of this decision problem is $\mathcal{O}(|\mathcal{T}|^2)$.

Concerning decision problem (b), one can easily see that $o \in \hat{\bar{I}}(s)$ if and only if all nodes that correspond to terms in s are reachable from the node that corresponds to object o. Thus, we can write $o \in \hat{\bar{I}}(s)$ iff $n_o \rightarrow^* n_t$, for each $t \in s$. As a consequence, the time complexity of this problem is $\mathcal{O}(|\mathcal{T}|^2|s|)$, where $|s|$ denotes the number of terms in s.

Consider now decision problem (c), i.e. $\overset{?}{\exists}$ o such that $o \in \hat{\bar{I}}(s)$. Notice that this is equivalent to the decision problem $s \overset{?}{\in} V(M)$. One method for deciding this problem is to run decision problem (b), for each $o \in Obj$. Thus, the time complexity of this problem is $\mathcal{O}(|Obj||\mathcal{T}|^2|s|)$.

Consider now decision problem (d), that is the compound ordering decision problem $s \overset{?}{\preceq} s'$, where $s = \{a_1, ..., a_k\}$ and $s' = \{b_1, ..., b_l\}$. According to Section 3.1, we have to perform at most $k*l$ term subsumption checks. If we have stored only the transitive reduction of the subsumption relation $\leq$, then term subsumption again reduces to the problem of reachability in the graph of the faceted taxonomy $\mathcal{F}$ (which is a subgraph of the source graph). As the terminologies of the facets are disjoint and subsumption links do not cross the boundaries of facets, each reachability has to process $|\mathcal{T}_i|^2$ edges. For simplicity, we can assume that $|\mathcal{T}_i| = |\mathcal{T}|/k$. Thus, the time complexity of deciding compound ordering is $\mathcal{O}((|\mathcal{T}|/k)^2|s||s'|)$.

In decision problem ($d_\circ$), we are interested in the complexity of $s \preceq s'$, when s, s' contain at most one term from each facet. As $|s|, |s'| \leq k$, the complexity of this task is $\mathcal{O}((|\mathcal{T}|/k)^2k^2) = \mathcal{O}(|\mathcal{T}|^2)$.

Task Id	Decision Task	Time Complexity
(a)	$o \overset{?}{\in} \bar{I}(t)$	$\mathcal{O}(\|\mathcal{T}\|^2)$
(b)	$o \overset{?}{\in} \hat{\bar{I}}(s)$	$\mathcal{O}(\|\mathcal{T}\|^2\|s\|)$
($b_\circ$)	$o \overset{?}{\in} \hat{\bar{I}}(s)$	$\mathcal{O}(\|\mathcal{T}\|^2 k)$
(c)	$\overset{?}{\exists}$ o such that $o \in \hat{\bar{I}}(s)$	$\mathcal{O}(\|Obj\|\|\mathcal{T}\|^2\|s\|)$
($c_\circ$)	$\overset{?}{\exists}$ o such that $o \in \hat{\bar{I}}(s)$	$\mathcal{O}(\|Obj\|\|\mathcal{T}\|^2 k)$
(d)	$s \overset{?}{\preceq} s'$	$\mathcal{O}((\|\mathcal{T}\|/k)^2\|s\|\|s'\|) = \mathcal{O}(\|\mathcal{T}\|^4/k^2)$
($d_\circ$)	$s \overset{?}{\preceq} s'$	$\mathcal{O}(\|\mathcal{T}\|^2)$

Table 3: Complexity of Some Basic Decision Problems

8.2 Complexity of Straightforward Expression Mining

Here, we will measure the computational complexity of the techniques described in Section 5. Recall that in this case we are interested only in the compound terms that contain at most one term from each facet, i.e. those in $V_\circ(M)$.

At first we will analyze the computational complexity of:

(f) computing $P := V(M) \cap G_{T_1,\ldots,T_k}$,

(g) computing $N := G_{T_1,\ldots,T_k} - V(M)$, and

(h) computing $minimal(P)$ (resp. $maximal(N)$).

Consider Task (f). This task requires checking whether $\hat{\bar{I}}(s) \neq \emptyset$ (i.e. decision problem ($c_\circ$)), for every compound term $s \in G_{T_1,\ldots,T_k}$. It follows that the complexity of Task (f) is $\mathcal{O}((|\mathcal{T}_1|+1) * \ldots * (|\mathcal{T}_k|+1)|Obj||\mathcal{T}|^2 k)$. Recall that according to the beginning of Section 8, $G_{T_1,\ldots,T_k}$ has at most $(1+\frac{|\mathcal{T}|}{k})^k$ elements. As a consequence, the complexity of Task (f) is $\mathcal{O}((\frac{|\mathcal{T}|}{k})^k|Obj||\mathcal{T}|^2 k) = \mathcal{O}(\frac{|\mathcal{T}|^{k+2}}{k^{k-1}}|Obj|)$. Note that for $k=1$ we get $\mathcal{O}(|\mathcal{T}|^3|Obj|)$.

Let us now consider Task (g), i.e. the task of computing $N := G_{T_1,\ldots,T_k} - V(M)$. The complexity of this task is the same with that of Task (f), as instead of selecting the valid, here we have to select the invalid compound terms for forming the set N.

Now consider Task (h), i.e. the task of computing the *minimal* (or the *maximal*) elements of a set A of compound terms. Clearly, we need to perform at most $|A|^2$ compound ordering checks (i.e. decision problem ($d_\circ$)). Thus, the time complexity for computing $minimal(P)$ (or $maximal(N)$) is $\mathcal{O}(|\mathcal{T}|^2|P|^2)$ (resp. $\mathcal{O}(|\mathcal{T}|^2|N|^2)$).

Let us now investigate how big $|P|$ (or $|N|$) can be. One can easily see that in the worst case $|P| = |G_{T_1,\ldots,T_k}|$, thus $|P| = \mathcal{O}(\frac{|\mathcal{T}|}{k})^k$. It follows that the complexity of computing $minimal(P)$ (or $maximal(N)$) is $\mathcal{O}(|\mathcal{T}|^2(\frac{|\mathcal{T}|}{k})^{2k}) = \mathcal{O}(\frac{|\mathcal{T}|^{2k+2}}{k^{2k}})$.

Now we are ready to describe the complexity of the straightforward techniques for expression mining that were described in Section 5. The complexity of Step 1 is $\mathcal{O}(\frac{|\mathcal{T}|^{k+2}}{k^{k-1}}|Obj|)$. The complexity of (the optional) Step 2 is $\mathcal{O}(\frac{|\mathcal{T}|^{2k+2}}{k^{2k}})$. So, the overall time complexity is: $\mathcal{O}(\frac{|\mathcal{T}|^{k+2}}{k^{k-1}}|Obj| + \frac{|\mathcal{T}|^{2k+2}}{k^{2k}})$. Table 4 summarizes the complexity results presented in this subsection.

At first glance, this complexity seems to be formidably high. However, note that k cannot practically be very big (we haven't seen so far any real application with more than 10 facets). Furthermore, $|\mathcal{T}|$, and even more $\frac{|\mathcal{T}|}{k}$, is not very big in a faceted taxonomy. For example, a faceted taxonomy of 8 facets each one having 10 terms (i.e. with $|\mathcal{T}| = 80$ and $\frac{|\mathcal{T}|}{k} = 10$) gives 100 million different compound terms. Note that this set of compound terms can discriminate 1 billion objects (e.g. Web pages) in blocks of 10 elements. Yet, the above straightforward techniques are quite naive and optimized algorithms with better performance are presented in Section 9.

Task Id	Task	Time Complexity
(f)	computing $P := V(M) \cap G_{T_1,...,T_k}$	$\mathcal{O}(\frac{\lvert\mathcal{T}\rvert^{k+2}}{k^{k-1}}\lvert Obj\rvert)$
(g)	computing $N := G_{T_1,...,T_k} - V(M)$	$\mathcal{O}(\frac{\lvert\mathcal{T}\rvert^{k+2}}{k^{k-1}}\lvert Obj\rvert)$
(h)	computing $minimal(P)$ (resp. $maximal(N)$)	$\mathcal{O}(\lvert\mathcal{T}\rvert^2\lvert P\rvert^2) = \mathcal{O}(\frac{\lvert\mathcal{T}\rvert^{2k+2}}{k^{2k}})$
(f/g and h)	straightforward expression mining	$\mathcal{O}(\frac{\lvert\mathcal{T}\rvert^{k+2}}{k^{k-1}}\lvert Obj\rvert + \frac{\lvert\mathcal{T}\rvert^{2k+2}}{k^{2k}})$

Table 4: Complexity of Straightforward Expression Mining

8.3 Complexity of Shortest Expression Mining

For measuring the computational complexity of the techniques described in Section 6, we start by analyzing the complexity of $SpecifyParams(e, M)$. First, we need to analyze the computational complexity of:

(i) computing $P := V_\circ(M) \cap G_{S_{e_1},...,S_{e_n}}$, and

(j) computing $N := G_{S_{e_1},...,S_{e_n}} - V_\circ(M)$.

Let us first introduce some notations. We shall use:
$f(t)$ to denote the facet to which the term t belongs to (e.g. $f(Crete) = Location$),
$f(s)$ to denote the set of facets of the terms in s (i.e. $f(s) = \{f(t) \mid t \in s\}$), and
$f(e)$ to denote the facets that appear in an expression e.
Using an appropriate encoding, the computation of $f(t)$ is in $\mathcal{O}(1)$, of $f(s)$ is in $\mathcal{O}(|s|)$, and of $f(e)$ is in $\mathcal{O}(k)$, where k is the number of facets of $\mathcal{F}$.

Let us now return to Task (i), i.e. to the task of computing $P := V_\circ(M) \cap G_{S_{e_1},...,S_{e_n}}$. For computing P we shall first compute $V_\circ(M)$ (i.e. $V(M) \cap (T_1 \oplus ... \oplus T_k)$), and then we will select those elements of $V_\circ(M)$ that also belong to $G_{S_{e_1},...,S_{e_n}}$.

Let $s \in V_\circ(M)$. From the construction of $S_{e_1}, ..., S_{e_n}$, one can see $s \in G_{S_{e_1},...,S_{e_n}}$ iff:
(1) there exist $i, j = 1, ..., n$, $i \neq j$, such that: $f(s) \cap f(e_i) \neq \emptyset$, $f(s) \cap f(e_j) \neq \emptyset$, and
(2) $f(s) \subseteq f(e)$.

Let us call Task (x), the task of deciding whether s satisfies conditions (1) and (2) above. For deciding this task, we first have to compute $f(s)$ and then take the intersection of $f(s)$ with every $f(e_i)$, where $i = 1, ..., n$ ($n \leq k$). If at least two of the above intersection operations yield a nonempty set, and $f(s) \subseteq f(e)$, then s satisfies conditions (1) and (2). As sets $f(s)$, $f(e_i)$, and $f(e)$ can have at most k elements ($\mathcal{F}$ contains k facets), Task (x) takes $\mathcal{O}(k^2)$ time. So, to decide whether an $s \in V_\circ(M)$ is also element of $G_{S_{e_1},...,S_{e_n}}$ requires $\mathcal{O}(k^2)$ time.

As a consequence, the overall time complexity of Task (i) is: $(\frac{|\mathcal{T}|}{k})^k * (Task(c_\circ) + Task(x)) = \mathcal{O}((\frac{|\mathcal{T}|}{k})^k(|Obj||\mathcal{T}|^2k + k^2)) = \mathcal{O}((\frac{|\mathcal{T}|}{k})^k(|Obj||\mathcal{T}|^2k))= \mathcal{O}(\frac{|\mathcal{T}|^{k+2}}{k^{k-1}}|Obj|)$.

Let us now consider Task (j), i.e. the task of computing $N := G_{S_{e_1},...,S_{e_n}} - V_\circ(M)$. Note that as $G_{S_{e_1},...,S_{e_n}}$ is subset of $G_{T_1,...,T_k}$, it holds[12]:

$$G_{S_{e_1},...,S_{e_n}} - V_\circ(M) = G_{S_{e_1},...,S_{e_n}} \cap (G_{T_1,...,T_k} - V_\circ(M))$$

This means that N can be computed as follows: for each s in $G_{T_1,...,T_k}$, we check if s is valid (i.e. if $s \in V_\circ(M)$). If not then we check if it belongs to $G_{S_{e_1},...,S_{e_n}}$ and if this is true, then we put s in N.

Let $s \in G_{T_1,...,T_k} - V_\circ(M)$. It follows from the definitions that $s \in G_{S_{e_1},...,S_{e_n}}$ iff:
(α) $\exists\, i, j = 1, ..., n,\ i \neq j$, such that: $f(s) \cap f(e_i) \neq \emptyset$ and $f(s) \cap f(e_j) \neq \emptyset$, and
(β) $s_{f(e_i)} \in S_{e_i}$ for each $i = 1, ..., n$, where $s_{f(e_i)} = \{t \in s \mid f(t) \in f(e_i)\}$.

The complexity of (α) is $\mathcal{O}(k^2)$ time (it is the Task (x) described earlier in this section). Concerning (β), notice that $s_{f(e_i)} \in S_{e_i}$ can hold only because $\hat{\bar{I}}(s_{f(e_i)}) \neq \emptyset$. So, for checking ($\beta$) we have to run n times ($n \leq k$) the decision problem ($c_\circ$). Thus, the time complexity of (β) is $\mathcal{O}(|Obj||\mathcal{T}|^2k^2)$. It follows that if $s \in G_{T_1,...,T_k} - V_\circ(M)$, the time complexity of deciding whether $s \in G_{S_{e_1},...,S_{e_n}}$ is $\mathcal{O}(|Obj||\mathcal{T}|^2k^2 + k^2) = \mathcal{O}(|Obj||\mathcal{T}|^2k^2)$.

As a consequence, the overall time complexity of Task (j) is: $(\frac{|\mathcal{T}|}{k})^k * (Task(c_\circ) + |Obj||\mathcal{T}|^2k^2)$ $= \mathcal{O}((\frac{|\mathcal{T}|}{k})^k|Obj||\mathcal{T}|^2k^2) = \mathcal{O}(\frac{|\mathcal{T}|^{k+2}}{k^{k-2}}|Obj|)$.

So, we can safely assume that the complexity of either Task(i) or Task (j) is $\mathcal{O}(\frac{|\mathcal{T}|^{k+2}}{k^{k-2}}|Obj|)$.

Let us now consider the algorithm $SpecifyParams(e, M)$. As each expression e has at most $k-1$ operations, we have to set $k-1$ parameter sets. For defining each such parameter set, we have to perform an operation of the form $P := V_\circ(M) \cap G_{S_{e_1},...,S_{e_n}}$ or $N := G_{S_{e_1},...,S_{e_n}} - V_\circ(M)$, and one $minimal(P)$ or $maximal(N)$ operation. So, the time complexity of $SpecifyParams(e, M)$ is:
$(k-1) * (Task(i) + Task(h)) = (k-1) * \mathcal{O}(\frac{|\mathcal{T}|^{k+2}}{k^{k-2}}|Obj| \ + \ \frac{|\mathcal{T}|^{2k+2}}{k^{2k}}) = \mathcal{O}(\frac{|\mathcal{T}|^{k+2}}{k^{k-3}}|Obj| \ + \ \frac{|\mathcal{T}|^{2k+2}}{k^{2k-1}})$.

Let us now consider the algorithm $FindShortestExpression(M)$. The number of parse trees returned by $ParseTrees(\{F_1, ..., F_k\})$ is at most $(k-1)!5^{k-1}$, where k is the number of facets (Prop. 1). So, the procedure $SpecifyParams(e, M)$ will be called at most $(k-1)!5^{k-1}$ times. Of course, we have to add the time complexity of the algorithm that finds all possible parse trees. However this task is computationally negligible, comparing it with the task of setting up the parameters. From this, we conclude that the complexity of $FindShortestExpression(M)$ is $(k-1)!5^{k-1}$ times greater than the complexity of $SpecifyParams(e, M)$, i.e.:

$$\mathcal{O}(\frac{|\mathcal{T}|^{k+2}}{k^{k-3}}(k-1)!5^k|Obj| \ + \ \frac{|\mathcal{T}|^{2k+2}}{k^{2k-1}}(k-1)!5^k)$$

Table 5 summarizes the complexity results presented in this subsection. From the remarks presented at the end of Section 8.2, it follows that the complexity of shortest expression mining is not formidably high. However, an optimized algorithm with much better complexity is going to be presented in the next section.

9 Optimizations

One can easily guess that expression mining is a computationally heavy task. In this section, we describe some more efficient algorithms for expression mining. In subsections 9.1 and 9.2,

[12] If A is subset of a set C then $A - B = A \cap B^C$, where B^C is the complement of B with respect to C.

Task Id	Task	Time Complexity
(i)	computing $P := V_\circ(M) \cap G_{S_{e_1},...,S_{e_n}}$	$\mathcal{O}(\frac{\lvert\mathcal{T}\rvert^{k+2}}{k^{k-2}}\lvert Obj\rvert)$
(j)	computing $N := G_{S_{e_1},...,S_{e_n}} - V_\circ(M)$	$\mathcal{O}(\frac{\lvert\mathcal{T}\rvert^{k+2}}{k^{k-2}}\lvert Obj\rvert)$
(sp)	$SpecifyParams$	$\mathcal{O}(\frac{\lvert\mathcal{T}\rvert^{k+2}}{k^{k-3}}\lvert Obj\rvert + \frac{\lvert\mathcal{T}\rvert^{2k+2}}{k^{2k-1}})$
(fse)	$FindShortestExpression$	$(k-1)!5^{k-1}$ * Task (sp)

Table 5: Complexity of Shortest Expression Mining

we give two optimized algorithms for mining a plus-product and a minus-product expression. In subsection 9.3, we show how we can reduce the number of parse trees that have to be investigated for finding the shortest expression. Finally, in subsection 9.4, we show how we can speed up the evaluation of the algorithm $SpecifyParams(e, M)$, by taking advantage of the properties of CTCA. At last, we will present an optimized algorithm for shortest expression mining that takes advantage of all particular optimizations.

9.1 Mining a Plus-Product Expression

In Section 5, we presented an expression mining algorithm for the case that e is a plus-product operation over $T_1, ..., T_k$, that is $e = \oplus_P(T_1, ..., T_k)$. Specifically, we showed that we can find the sought parameter P by going through each $s \in G_{T_1,...,T_k}$, and then checking whether $\hat{\hat{I}}(s) \neq \emptyset$. Parameter P is set equal to the minimal elements of the collected compound terms, that is $P = minimal(V(M) \cap G_{T_1,...,T_k})$. In this subsection, we will present a more efficient algorithm, called $PlusMining(M)$, for computing the parameter P.

First, we will give a few auxiliary definitions. Given an object $o \in Obj$, we shall use $I^{-1}(o)$ to denote the set of terms t such that $o \in I(t)$, i.e. $I^{-1}(o) = \{t \in \mathcal{T} \mid o \in I(t)\}$. Intuitively, $I^{-1}(o)$ is the valid compound term corresponding to object o. Now, collecting all these valid compound terms, we define $D_I = \{I^{-1}(o) \mid o \in Obj\}$. It is easy to see that $Br(D_I) = V(M)$.

Now, as we are interested in $V_\circ(M)$, we define

$$D'_I = \{tupleToSet(\pi_1(s) \times ... \times \pi_k(s)) \mid s \in D_I\}, \text{ where}$$

$\pi_i(s)$ is the set of terms of s that belong to facet F_i, and $tupleToSet$ is a function that transforms each tuple $(t_1, ..., t_n)$ to the set $\{t_1, ..., t_n\}$[13]. Intuitively, D'_I is the set of valid compound terms derived from D_I that consist of at most one term from each facet and have more than one term (i.e., that belong to $G_{T_1,...,T_k}$). It is easy to see that the following proposition holds:

Prop. 2 Let M be a materialized faceted taxonomy. It holds:
(i) $Br(D'_I) = V(M) \cap G_{T_1,...,T_k}$, and
(ii) $minimal(D'_I) = minimal(V(M) \cap G_{T_1,...,T_k})$.

This last result gives us a simple, yet efficient algorithm for computing P such that $S_{\oplus_P(T_1,...,T_k)} = V_\circ(M)$. In particular, the algorithm $PlusMining(M)$ (Alg. 9.1) takes as input a materialized faceted taxonomy M, computes D'_I, and returns the desired parameter $P = minimal(D'_I)$. In the algorithm, symbols I_o^{-1}, D, and D' correspond to $I^{-1}(o)$, D_I, and D'_I, respectively.

Let us now give an example of $PlusMining(M)$ for the materialized faceted taxonomy shown in Figure 5. Considering object 4 ($o4$), $I_{o4}^{-1} = \{Cefalonia, Seaski, Windsurfing\}$. Considering

[13] If $\pi_i(s) = \emptyset$, for an $i = 1, ..., k$, then $\pi_i(s)$ is excluded from $\pi_1(s) \times ... \times \pi_k(s)$.

Algorithm 9.1 $PlusMining(M)$
Input: A materialized faceted taxonomy $M = (\mathcal{F}, I)$
Output: A set of compound terms P such that $S_{\oplus_P(T_1,...,T_k)} = V_\circ(M)$

(1) $D := \emptyset$;
(2) For each o in Obj do
(3) $\quad I_o^{-1} := \{t \in \mathcal{T} \mid o \in I(t)\}$;
(4) $\quad D := D \cup \{I_o^{-1}\}$;
(5) EndFor
(6) $D' := \{tupleToSet(\pi_1(s) \times ... \times \pi_k(s)) \mid s \in D\}$;
(7) $P := minimal(D')$;
(8) return P

all objects, we derive $D = \{\{Olympus, Snowski\}, \{Crete, SeaSki\}, \{Crete, Windsurfing\}, \{Cefalonia, Seaski, Windsurfing\}\}$. Actually, $Br(D) = V(M)$. However, as we are interested in $V_\circ(M)$, through line (6) we get the valid compound terms that consist of at most one term from each facet. Specifically, we get $D' = \{\{Olympus, Snowski\}, \{Crete, SeaSki\}, \{Crete, Windsurfing\}, \{Cefalonia, Seaski\}, \{Cefalonia, Windsurfing\}\}$[14]. In this example, $D' = minimal(D')$. Therefore, the returned parameter P is the same as D'.

Let us compute the time complexity of algorithm $PlusMining(M)$. The time complexity of computing I_o^{-1} is $\mathcal{O}(|\mathcal{T}|)$, as object o may be associated with at most $|\mathcal{T}|$ terms. Thus, to compute D takes $\mathcal{O}(|Obj| * |\mathcal{T}|)$ time. As $|D| \leq |Obj|$ and $|s| \leq |\mathcal{T}|$ for $s \in D$, computing $\pi_1(s)$, ..., $\pi_k(s)$ for all $s \in D$ takes $\mathcal{O}(|Obj||\mathcal{T}|)$ time. Thus, the cost of line (6) is $\mathcal{O}(|Obj||\mathcal{T}| + |D'|)$. The cost of computing $minimal(D')$ is $\mathcal{O}(|D'|^2|\mathcal{T}|^2)$ (see Task (h)). Thus, the overall cost of algorithm $PlusMining(M)$ is:

$$Task(pm) = \mathcal{O}(|Obj||\mathcal{T}| + |D'_I|^2|\mathcal{T}|^2)$$

As in practice $|D'_I| << |V(M) \cap G_{T_1,...,T_k}|$, it follows that algorithm $PlusMining(M)$ is much more efficient than the straightforward one of Section 5.

9.2 Mining a Minus-Product Expression

In this subsection, we focus on the case that e is a minus-product operation over $T_1, ..., T_k$, that is $e = \ominus_N(T_1, ..., T_k)$. As we showed in Section 5, we can find the sought parameter N by going through each $s \in G_{T_1,...,T_k}$, and then checking whether $\hat{\bar{I}}(s) = \emptyset$. The parameter N is set equal to the maximal elements of the collected compound terms, that is $N = maximal(G_{T_1,...,T_k} - V(M))$.

However, several optimizations to this (quite naive) algorithm are possible. For instance, an optimized algorithm when realizes that a compound term, say $\{a, b\}$, is invalid, it does not proceed in checking the compound terms that contain $\{a, b\}$, e.g. $\{a, b, c\}$, because $\{a, b, c\}$ is certainly invalid and not broader (w.r.t. $\preceq$) than $\{a, b\}$. Such an optimization is performed in the algorithm $MinusMining(M)$ (Alg. 9.2)[15]. In the algorithm, D collects selectively invalid compound terms by taking into account the previous optimization. Thus, $D \subseteq G_{T_1,...,T_k} - V(M)$, and $maximal(D) = maximal(G_{T_1,...,T_k} - V(M))$. The algorithm returns the desired parameter $N = maximal(D)$.

[14]Note that $\pi_1(\{Cefalonia, Seaski, Windsurfing\}) = \{Cefalonia\}$ and $\pi_2(\{Cefalonia, Seaski, Windsurfing\}) = \{Seaski, Windsurfing\}$.

[15]Note that each $s_i \in T_i, i = 1, ..., k$, is singleton or $\emptyset$. We set $\hat{\bar{I}}(\emptyset)$ equal to a special value OBJ such that $OBJ \neq \emptyset$ and $R_i \cap OBJ = R_i$.

Algorithm 9.2 $MinusMining(M)$
Input: A materialized faceted taxonomy $M = (\mathcal{F}, I)$
Output: A set of compound terms N such that $S_{\ominus_N(T_1,...,T_k)} = V_\circ(M)$

$D := \emptyset$;
For each $s_1 \in T_1$ do
  $R_1 := \hat{\tilde{I}}(s_1)$;
  If $R_1 \neq \emptyset$ then
    For each $s_2 \in T_2$ do
      $R_2 := R_1 \cap \hat{\tilde{I}}(s_2)$;
      If $R_2 \neq \emptyset$ then
        ...
        If $R_{k-1} \neq \emptyset$ then
          For each $s_k \in T_k$ do
            $R_k := R_{k-1} \cap \hat{\tilde{I}}(s_k)$;
            If $R_k = \emptyset$ then $D := D \cup \{s_1 \cup ... \cup s_k\}$;
          End for
        Else $D := D \cup \{s_1 \cup ... \cup s_{k-1}\}$;
        ...
      Else $D := D \cup \{s_1 \cup s_2\}$;
    End for
End for
$N := maximal(D)$;
return(N)

In practice, the algorithm $MinusMining(M)$ is much more efficient than the exhaustive one of Section 5. However both algorithms have the same worst case complexity.

Other optimizations are also possible. Specifically, we can exploit the subsumption relation $\leq$ in order to avoid checking the validity of some compound terms. For example, if $a2 < a1$ and $\{a1, b\}$ is invalid then there is no need to check the compound term $\{a2, b\}$, because it is certainly invalid and not broader (w.r.t. $\preceq$) than $\{a1, b\}$.

9.3 Reducing the Number of Candidate Parse Trees

In this subsection, we show that some parse trees returned by $ParseTrees(\{F_1, ..., F_k\})$ can been eliminated before $SpecifyParams(e, M)$ is called, for setting up the parameters. This is because, these parse trees cannot yield to the shortest expression. We start by some auxiliary definitions.

Def 9.1 We say that two well-formed expressions e and e' are *equivalent*, and write $e \equiv e'$, iff $S_e = S_{e'}$.

Def 9.2 A well-formed expression e is *space-minimal* if every P and N parameter of e satisfies the equalities $P = minimal(P)$ and $N = maximal(N)$. We shall call *space-redundant* every well-formed expression that is not space-minimal.

Clearly, we can always convert a space-redundant expression to a space-minimal expression. Also notice that all algorithms that we have described so far yield space-minimal expressions. Indeed, we are interested only in space-minimal expressions.

Prop. 3 Let e and e' be two space-minimal and equivalent (i.e. $e \equiv e'$) expressions. If e has the form $e = \oplus_P(e_1, ..., e_n)$, for $n \geq 3$, and

$$\begin{aligned} e' &= \oplus_{P1}(\oplus_{P2}(e_1, ..., e_{n-1}), e_n) \text{ , or} \\ e' &= \oplus_{P1}(\oplus_{P2}(e_1, ..., e_{l-1}), \oplus_{P3}(e_l, ..., e_n)), \text{ where } 2 < l < n \end{aligned}$$

then it holds $size(e) \leq size(e')$.

Prop. 4 Let e and e' be two space-minimal and equivalent (i.e. $e \equiv e'$) expressions. If e has the form $e = \ominus_N(e_1, ..., e_n)$, for $n \geq 3$, and

$$\begin{aligned} e' &= \ominus_{N1}(\ominus_{N2}(e_1, ..., e_{n-1}), e_n) \text{ , or} \\ e' &= \ominus_{N1}(\ominus_{N2}(e_1, ..., e_{l-1}), \ominus_{N3}(e_l, ..., e_n)), \text{ where } 2 < l < n \end{aligned}$$

then it holds $size(e) \leq size(e')$.

The above two propositions allow us to exclude several parse trees during shortest expression mining. For example, we can exclude the parse trees tr_4, tr_6, tr_8, tr_{12}, tr_{14}, and tr_{16} returned by $ParseTrees(\{A, B, C\})$ (Figure 6(c)) from the candidates, because the size of their corresponding expression can never be smaller than that of the expressions of the remaining parse trees.

For skipping these parse trees, we have to modify $FindShortestExpression(M)$ as follows: before calling (at line (4) of Alg. 6.1) the computationally expensive $SpecifyParams(e, M)$ for a parse tree e, we first check if e is a "good candidate". We can check whether a parse tree e is a "good candidate" by a simple algorithm: First, we traverse the parse tree e until we reach an internal node n with a label l, such that (i) n has a child which is also an internal node, and (ii) all of the children of n which are internal nodes have the same label l. If we reach such a node n of e, we conclude that the parse tree e is not a "good candidate". In that case, we don't proceed in calling $SpecifyParams(e, M)$.

Hopefully, more than half of the parse trees returned by $ParseTrees(\{F_1, ..., F_k\})$ are redundant, and thus can be ignored. For example, from the 162 parse trees returned by $ParseTrees(\{A, B, C, D\})$ only 64 are good candidates. Another important remark is that as efficiently computable approximate optimal solutions would be more than acceptable for this problem case, it seems that the employment of genetic algorithms [11] would fit quite well to this problem in order to further reduce the number of parse trees. This is an issue for further research and experimental evaluation.

9.4 Speeding up the Specification of Parameters in Shortest Expression Mining

In this section, we present an optimized version of the algorithm $FindShortestExpression(M)$ (Alg. 6.1). The optimized algorithm exploits some algebraic properties of CTCA and it is also enhanced with the optimizations described in the previous subsections.

At first, we need to introduce some definitions. Let M be a materialized faceted taxonomy, we define:

$$\begin{aligned} V_\circ^+(M) &= minimal(V_\circ(M) \cap G_{T_1,...,T_k}), \text{and} \\ V_\circ^-(M) &= maximal(G_{T_1,...,T_k} - V_\circ(M)) \end{aligned}$$

Figure 9 illustrates graphically the relationships between the sets $V_\circ^+(M)$ and $V_\circ^-(M)$ and the sets $V_\circ(M)$ and $G_{T_1,...,T_k} - V_\circ(M)$. Actually, $V_\circ^+(M)$ and $V_\circ^-(M)$ are the sets P and N of the straightforward expression mining techniques that were described in Section 5. Therefore, $V_\circ^+(M) = PlusMining(M)$, and $V_\circ^-(M) = MinusMining(M)$.

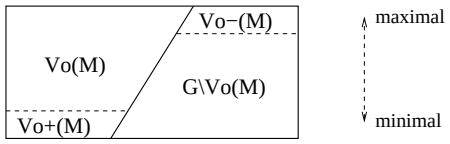

Figure 9: The set relationships of $V_\circ^+(M)$ and $V_\circ^-(M)$.

Our optimization idea is to replace the operations $minimal(G_{S_{e_1},...,S_{e_n}} \cap V_\circ(M))$ and $maximal(G_{S_{e_1},...,S_{e_n}} - V_\circ(M))$ performed in algorithm $SpecifyParams(e, M)$ (Alg. 6.2), with equivalent but more efficient operations that use the sets $V_\circ^+(M)$ and $V_\circ^-(M)$ (instead of the much larger ones $V_\circ(M)$ and $G_{T_1,...,T_k} - V_\circ(M)$). Indeed, we will show that such an optimization is possible.

The proposed optimization uses the following auxiliary definition. Let e be a well-formed expression, and let $S \subseteq P(\mathcal{T})$. We define: $\pi_e(S) = \{s_{f(e)} \mid s \in S\}$[16]. Intuitively, $\pi_e(S)$ is the "projection" of the compound terms in S to the facets appearing in e.

The following proposition holds[17]:

Prop. 5 Let M be a materialized faceted taxonomy, and let e_i, $i = 1, .., n$, be well-formed expressions such that $f(e_i) \cap f(e_j) = \emptyset$, for $i \neq j$. Let $S_{e_i} = V_\circ(M) \cap P(\mathcal{T}_{e_i})$, for $i = 1, ..., n$. It holds:

$$minimal(G_{S_{e_1},...,S_{e_n}} \cap V_\circ(M)) = minimal(G_{S_{e_1},...,S_{e_n}} \cap \pi_e(V_\circ^+(M))),$$
where $e = \oplus_P(e_1, ..., e_n)$ or $e = \ominus_N(e_1, ..., e_n)$

Obviously, the operation $minimal(G_{S_{e_1},...,S_{e_n}} \cap \pi_e(V_\circ^+(M)))$ is more efficient than the operation $minimal(G_{S_{e_1},...,S_{e_n}} \cap V_\circ(M))$, because the sets $V_\circ^+(M)$ and $\pi_e(V_\circ^+(M))$ are much smaller than $V_\circ(M)$, and $G_{S_{e_1},...,S_{e_n}} \cap \pi_e(V_\circ^+(M))$ is much smaller than $G_{S_{e_1},...,S_{e_n}} \cap V_\circ(M)$.

Additionally, the following proposition holds:

Prop. 6 Let M be a materialized faceted taxonomy, and let e_i, $i = 1, .., n$, be well-formed expressions such that $f(e_i) \cap f(e_j) = \emptyset$, for $i \neq j$. Let $S_{e_i} = V_\circ(M) \cap P(\mathcal{T}_{e_i})$, for $i = 1, ..., n$. It holds:

$$maximal(G_{S_{e_1},...,S_{e_n}} - V_\circ(M)) = G_{S_{e_1},...,S_{e_n}} \cap V_\circ^-(M)$$

Obviously, the operation $G_{S_{e_1},...,S_{e_n}} \cap V_\circ^-(M)$ is more efficient than the operation $maximal(G_{S_{e_1},...,S_{e_n}} - V_\circ(M)) = maximal(G_{S_{e_1},...,S_{e_n}} \cap (G_{T_1,...,T_k} - V_\circ(M)))$, as $V_\circ^-(M)$ is much smaller than $G_{T_1,...,T_k} - V_\circ(M)$ and no $maximal$ operation is needed.

Based on the above propositions, we can reduce the computational cost of the algorithm $SpecifyParams(e, M)$ (Alg. 6.2) by replacing the operations $minimal(G_{S_{e_1},...,S_{e_n}} \cap V_\circ(M))$ and $maximal(G_{S_{e_1},...,S_{e_n}} - V_\circ(M))$ with their equivalent, but more efficient operations $minimal(G_{S_{e_1},...,S_{e_n}} \cap \pi_e(V_\circ^+(M)))$ and $G_{S_{e_1},...,S_{e_n}} \cap V_\circ^-(M)$.

We are now ready to give the optimized algorithm $SpecifyParamsO(e, V^+, V^-)$, where $V^+ = V_\circ^+(M)$ and $V^- = V_\circ^-(M)$.

[16]Recall that $s_{f(e)} = \{t \in s \mid f(t) \in f(e)\}$, where $f(t)$ is the facet of term t and $f(e)$ are the facets appearing in expression e.

[17]Recall that $\mathcal{T}_e$ is the union of the terminologies appearing in expression e.

Algorithm 9.3 $SpecifyParamsO(e, V^+, V^-)$
Input: The parse tree of an expression e, the set $V^+ = V_\circ^+(M)$, and the set $V^- = V_\circ^-(M)$
Output: The parse tree of e enriched with P and N parameters, such that $S_e = V_\circ(M)$ and e is space-minimal

(1) case(e) {
(2) $\oplus_P(e_1, ..., e_n)$: For $i := 1, ..., n$ do
(3) $e_i :=$ **SpecifyParamsO**(e_i, V^+, V^-) ;
(4) $P' := G_{S_{e_1}, ..., S_{e_n}} \cap \pi_e(V^+)$;
(5) $e.P := minimal(P')$;
(6) return(e)
(7) $\ominus_N(e_1, ..., e_n)$: For $i := 1, ..., n$ do
(8) $e_i :=$ **SpecifyParamsO**(e_i, V^+, V^-) ;
(9) $e.N := G_{S_{e_1}, ..., S_{e_n}} \cap V^-$;
(10) return(e)
(11) T_i: return(e)
(12) }

Below, we give the optimized algorithm $FindShortestExpressionO(M)$ for shortest expression mining (Alg. 9.4). The algorithm first computes the parameters $V^+ = V_\circ^+(M)$ and $V^- = V_\circ^-(M)$, using the optimized algorithms $PlusMining(M)$ (Alg. 9.1) and $MinusMining(M)$ (Alg. 9.2). Then, it calls the procedure $SpecifyParamsO(e, V^+, V^-)$, for each parse tree $e \in ParseTrees(\{F_1, ..., F_k\})$ which is a "good candidate", and not a plus-product or a minus-product operation over $T_1, ..., T_k$.

Algorithm 9.4 $FindShortestExpressionO(M)$
Input: A materialized faceted taxonomy M with facets $\{F_1, ..., F_k\}$
Output: The shortest expression e such that $S_e = V_\circ(M)$

$V^+ :=$ **PlusMining**(M); // i.e., $V^+ = V_\circ^+(M)$
$V^- :=$ **MinusMining**(M); // i.e, $V^- = V_\circ^-(M)$
$minSize :=$ MAXINT;
$shortestExpr :=$ "";
For each e in **ParseTrees**$(\{F_1, ..., F_k\})$ and **GoodCandidate**(e) do
  if $e \neq \oplus_P(T_1, ..., T_k)$ or $e \neq \ominus_N(T_1, ..., T_k)$ then $e' :=$ **SpecifyParamsO**(e, V^+, V^-);
  if $e = \oplus_P(T_1, ..., T_k)$ then $e' := \oplus_{V^+}(T_1, ..., T_k)$;
  if $e = \ominus_N(T_1, ..., T_k)$ then $e' := \ominus_{V^-}(T_1, ..., T_k)$;
  $size :=$ **GetSize**(e');
  If $size < minSize$ then
    $minSize := size$;
    $shortestExpr := e'$;
  EndIf
EndFor
return ($shortestExpr$)

Note that if the parse tree e has the form $\oplus_P(T_1, ..., T_k)$ then there is no need to call $SpecifyParamsO(e, V^+, V^-)$ because P should be set equal to $V_\circ^+(M)$, which is actually the already computed parameter V^-. Similarly, if the parse tree e has the form $\ominus_N(T_1, ..., T_k)$ then there is no need to call $SpecifyParamsO(e, V^+, V^-)$, as N should be set equal to $V_\circ^-(M)$, which is the already computed parameter V^-.

Thus, the computation of V^+ and V^- in $FindShortestExpressionO(M)$ does not add any extra cost with respect to $FindShortestExpression(M)$, as the cost of computing V^+ and V^- is

smaller than the cost of $SpecifyParams(\oplus_P(T_1,...,T_k),M)$ and $SpecifyParams(\ominus_N(T_1,...,T_k),M)$, called by $FindShortestExpression(M)$. Note that, $FindShortestExpressionO(M)$ wisely does not call $SpecifyParamsO(\oplus_P(T_1,...,T_k),V^+,V^-)$ and $SpecifyParamsO(\ominus_N(T_1,...,T_k),V^+,V^-)$ at all.

Summarizing, the algorithm $FindShortestExpressionO(M)$ is an optimization of the algorithm $FindShortestExpression(M)$ with respect to several aspects:

- it uses the optimized algorithms $PlusMining(M)$ and $MinusMining(M)$, for computing $V_\circ^+(M)$ and $V_\circ^-(M)$, respectively,
- it examines only the parse trees that are "good candidates" for the shortest expression, and
- it calls the optimized algorithm $SpecifyParamsO(e,V^+,V^-)$, instead of $SpecifyParams(e,M)$.

Below, we compute the worst-case complexity of $FindShortestExpressionO(M)$, and we compare it with that of $FindShortestExpression(M)$.

Let us fist compute the complexity of $SpecifyParamsO(e,V^+,V^-)$, where $V^+ = V_\circ^+(M)$ and $V^- = V_\circ^-(M)$. We will compute the complexity of the computation $minimal(G_{S_{e_1},...,S_{e_n}} \cap \pi_e(V^+))$. The complexity of computing $\pi_e(V^+)$ is $\mathcal{O}(|V^+||s||f(e)|)$ $=\mathcal{O}(|V^+|k^2)$. Let $s \in \pi_e(V^+)$. From the construction of $S_{e_1},...,S_{e_n}$, one can see that $s \in G_{S_{e_1},...,S_{e_n}}$ iff:
there exist $i,j=1,...,n,\ i \neq j$, such that: $f(s) \cap f(e_i) \neq \emptyset$, and $f(s) \cap f(e_j) \neq \emptyset$.
As sets $f(s)$, $f(e_i)$, for $i=1,...,n$, have at most k elements, the complexity of the check $s \in G_{S_{e_1},...,S_{e_n}}$ is $\mathcal{O}(k^2)$. Thus, the overall complexity of the computation $G_{S_{e_1},...,S_{e_n}} \cap \pi_e(V^+)$ is $\mathcal{O}(|V^+|k^2)$. The complexity of the computation $minimal(P')$ (line (5) of Alg. 9.3) is $\mathcal{O}(|P'|^2|\mathcal{T}|^2)$ $=\mathcal{O}(|V^+|^2|\mathcal{T}|^2)$ (see Task (h)). Therefore, the overall complexity of the operation $minimal(G_{S_{e_1},...,S_{e_n}} \cap \pi_e(V^+))$ is $\mathcal{O}(|V^+|^2|\mathcal{T}|^2)$.

Now, we will compute the complexity of the computation $G_{S_{e_1},...,S_{e_n}} \cap V^-$. Let $s \in V^-$. From the construction of $S_{e_1},...,S_{e_n}$, one can see that $s \in G_{S_{e_1},...,S_{e_n}}$ iff:
(α) $\exists\, i,j=1,...,n,\ i \neq j$, such that: $f(s) \cap f(e_i) \neq \emptyset$ and $f(s) \cap f(e_j) \neq \emptyset$, and
(β) $s_{f(e_i)} \in S_{e_i}$ for each $i=1,...,n$, where $s_{f(e_i)} = \{t \in s \mid f(t) \in f(e_i)\}$.

The complexity of (α) is $\mathcal{O}(k^2)$ time. Concerning (β), notice that $s_{f(e_i)} \in S_{e_i}$ can hold only because $\hat{\tilde{I}}(s_{f(e_i)}) \neq \emptyset$. So, for checking (β) we have to run n times ($n \leq k$) the decision problem ($c_\circ$). Thus, the time complexity of (β) is $\mathcal{O}(|Obj||\mathcal{T}|^2k^2)$.

Therefore, the complexity of the computation $G_{S_{e_1},...,S_{e_n}} \cap V^-$ is $\mathcal{O}(|V^-||Obj||\mathcal{T}|^2k^2)$.

As the above computations are called at most $(k-1)$ times, the overall complexity of $SpecifyParamsO(e,V^+,V^-)$ is:

$$Task(spo) \;=\; \mathcal{O}(|V_\circ^+(M)|^2|\mathcal{T}|^2k + |V_\circ^-(M)||Obj||\mathcal{T}|^2k^3)$$

Let us now compute the computational complexity of $FindShortestExpressionO(M)$. The complexity of computing V^+ and V^- is the same as the complexity of $PlusMining(M)$ and $MinusMining(M)$, respectively, that is (in the worst case):

$$\mathcal{O}(\frac{|\mathcal{T}|^{k+2}}{k^{k-1}}|Obj| \;+\; \frac{|\mathcal{T}|^{2k+2}}{k^{2k}})$$

Additionally, note that the procedure $SpecifyParamsO(e,V^+,V^-)$ is called at most PT_k times, where PT_k is the number of parse trees returned by $ParseTrees(\{F_1,...,F_k\})$[18]. From

[18]Recall that, only parse trees that are "good candidates" are examined.

Prop. 1, $PT_k \leq (k-1)! * 5^{k-1}$. Thus, the overall complexity of $FindShortestExpressionO(M)$ is:

$$\mathcal{O}(\frac{|\mathcal{T}|^{k+2}}{k^{k-1}}|Obj| \ + \ \frac{|\mathcal{T}|^{2k+2}}{k^{2k}} \ + \ (k-1)!5^k * Task(spo))$$

Comparing the worst-case complexity of $FindShortestExpressionO(M)$ with that of $FindShortestExpression(M)$ (given in Table 5), we get that the improvement of the former over the latter is

$$\Theta((k-1)!5^k * (\frac{|\mathcal{T}|^{k+2}}{k^{k-3}}|Obj| \ + \ \frac{|\mathcal{T}|^{2k+2}}{k^{2k-1}} - |V_\circ^+(M)|^2|\mathcal{T}|^2k - |V_\circ^-(M)||Obj||\mathcal{T}|^2k^3))$$

As in practice $|V_\circ^+(M)|$, $|V_\circ^-(M)| << \frac{|\mathcal{T}|^k}{k^k}$, the improvement of the worst-case complexity of $FindShortestExpressionO(M)$ over that of $FindShortestExpression(M)$ is obvious. Moreover, recall that $FindShortestExpressionO(M)$ employs several optimizations which contribute to its efficiency.

10 Conclusion

Mining commonly seeks for statistically justified associations or rules. In this paper, we deal with a different kind of mining. Specifically, we show how from an object base we can mine the associations of the elements of the schema (here of the terms of a faceted taxonomy) that capture information about the knowledge domain and were never expressed explicitly. Now, as the number of associations of this kind can be very big, we employ the Compound Term Composition Algebra (CTCA) for encoding and representing these associations compactly.

In particular, we studied *expression mining* in materialized faceted taxonomies, i.e. the problem of finding algebraic expressions of CTCA that specify exactly those compound terms that are extensionally valid (i.e. have non-empty interpretation) in a materialized faceted taxonomy. Applications of expression mining include: reusing the knowledge of materialized faceted taxonomies, reorganizing single-hierarchical taxonomies to faceted taxonomies, optimizing query answering, language engineering, recommending (or error avoiding) services, and optimizing user-defined algebraic expressions.

First, we considered the case that we are interested in the valid compound terms that consist of at most one term from each facet. For this case, we gave two straightforward methods for mining a plus-product and a minus-product expression (possibly, none the shortest), and an exhaustive algorithm for finding the shortest expression. The last algorithm yields expressions with remarkably low storage space requirements, thanks to the closed-world assumptions of CTCA. Moreover, we presented optimizations for all of the above mining cases.

We analyzed the computational complexity of all algorithms and reported their worst-case time complexity. We found out that though the unoptimized algorithms are computationally heavy, their optimized versions are quite efficient in practice. Of course, the complexity of shortest expression mining is much higher than the complexity of mining a plus-product or a minus-product expression due to the fact that the number of expressions that need to be examined for finding the shortest one is exponential with respect to the number of facets. This does not reduce the benefits of our approach, as the number of facets cannot practically be very big (we haven't seen so far any faceted taxonomy with more than 10 facets), and expression mining is a rare off-line task. As explained in the paper, the time for checking compound term validity is proportional to expression size. Thus, we considered that slow runs of shortest

expression mining can be tolerated in order to minimize the size of the mined expression and provide efficiency for later on-line tasks, such as object indexing and navigation.

We also presented an algorithm for shortest expression mining, in the case that we are interested in all valid compound terms. However, the complexity of shortest expression mining in this most general case is expected to be significantly higher (this is an issue for further research).

Finally, let us compare the proposed technique with the existing compression algorithms. Commonly, lossless compression algorithms are efficient for files that contain lots of repetitive data, i.e. sequences of data that occur more than once. Such algorithms range from the simple RLE (Run Length Encoding) algorithm to statistical algorithms, like Huffman compression [12], and dictionary-based algorithms, like LZ77 [33] (a variant of which is Gzip) and LZ78 [34]. Roughly, statistical algorithms assign a smaller number of bits to symbols with higher probability of appearance, while dictionary based algorithms substitute a sequence of symbols by a pointer to a previous occurrence of that sequence. The key difference of the technique proposed in this paper is that it exploits the semantics of the data, so it can reduce the size of a set of compound terms even if there are no repetitive data. It is evident that this kind of compression cannot be obtained by the classical (general purpose) compression algorithms. However, the classical compression algorithms could be applied as a post-processing step in order to further reduce the storage space. For instance, all facet terms can be stored in a dictionary form, i.e. a code can be assigned to each term and these codes can then be used (instead of the terms) for storing the P/N parameters.

References

[1] "XFML: eXchangeable Faceted Metadata Language". http://www.xfml.org.

[2] "XFML+CAMEL:Compound term composition Algebraically-Motivated Expression Language". http://www.csi.forth.gr/markup/xfml+camel.

[3] H. H. Bock and E. Diday. *Analysis of Symbolic Data*. Springer-Verlag, 2000. ISBN: 3-540-66619-2.

[4] Peter Clark, John Thompson, Heather Holmback, and Lisbeth Duncan. "Exploiting a Thesaurus-based Semantic Net for Knowledge-based Search". In *Procs of 12th Conf. on Innovative Applications of AI (AAAI/IAAI'00)*, pages 988–995, 2000.

[5] Princeton University Cognitive Science Laboratory. "WordNet: A Lexical Database for the English Language". (http://www.cogsci.princeton.edu/ wn).

[6] Edwin Diday. "An Introduction to Symbolic Data Analysis and the Sodas Software". *Electronic Journal of Symbolic Data Analysis*, 0(0), 2002.

[7] F. M. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. "Reasoning in Description Logics". In Gerhard Brewka, editor, *Principles of Knowledge Representation*, chapter 1, pages 191–236. CSLI Publications, 1996.

[8] Elizabeth B. Duncan. "A Faceted Approach to Hypertext". In Ray McAleese, editor, *HYPERTEXT: theory into practice, BSP*, pages 157–163, 1989.

[9] Hatem Haddad. "French Noun Phrase Indexing and Mining for an Information Retrieval System.". In *International Symposium on String Processing and Information Retrieval (SPIRE 2003)*, Manaus, Brasil, October 2003.

[10] David J. Hand, Heikki Mannila, and Padhraic Smyth. *Principles of Data Mining*. MIT Press, 2001. ISBN: 1-57735-027-8.

[11] John H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. MIT Press, Cambridge, MA, 1992. ISBN: 0262581116.

[12] D. Huffman. "A Method for the Construction of Minimum-Redundancy Codes". *Proc. of the Institute of Radio Engineers (IRE)*, 40(9):1098–1101, 1952.

[13] International Organization For Standardization. "Documentation - Guidelines for the establishment and development of monolingual thesauri", 1986. Ref. No ISO 2788-1986.

[14] P. H. Lindsay and D. A. Norman. *Human Information Processing.* Academic press, New York, 1977.

[15] Amanda Maple. "Faceted Access: A Review of the Literature", 1995. http://theme.music.indiana.edu/tech_s/mla/facacc.rev.

[16] C.H. Papadimitriou. *Computational complexity.* Addison-Wesley, 1994.

[17] Ruben Prieto-Diaz. "Classification of Reusable Modules". In *Software Reusability. Volume I*, chapter 4, pages 99–123. acm press, 1989.

[18] Ruben Prieto-Diaz. "Implementing Faceted Classification for Software Reuse". *Communications of the ACM*, 34(5):88–97, 1991.

[19] U. Priss and E. Jacob. "Utilizing Faceted Structures for Information Systems Design". In *Proceedings of the ASIS Annual Conf. on Knowledge: Creation, Organization, and Use (ASIS'99)*, October 1999.

[20] James Pustejovsky. *The Generative Lexicon.* MIT Press, Cambridge, 1995. ISBN: 0-262-16158-3.

[21] S. R. Ranganathan. "The Colon Classification". In Susan Artandi, editor, *Vol IV of the Rutgers Series on Systems for the Intellectual Organization of Information.* New Brunswick, NJ: Graduate School of Library Science, Rutgers University, 1965.

[22] Nicolas Spyratos, Yannis Tzitzikas, and Vassilis Christophides. "On Personalizing the Catalogs of Web Portals". In *15th International FLAIRS Conference, FLAIRS'02*, pages 430–434, Pensacola, Florida, May 2002.

[23] Yannis Tzitzikas. "An Algebraic Method for Compressing Very Large Symbolic Data Tables". In *Procs. of the Workshop on Symbolic and Spatial Data Analysis of ECML/PKDD 2004*, Pisa, Italy, September 2004.

[24] Yannis Tzitzikas and Anastasia Analyti. "Mining the Meaningful Compound Terms from Materialized Faceted Taxonomies". In *Procs. of the 3rd Intern. Conference on Ontologies, Databases and Applications of Semantics for Large Scale Information Systems, ODBASE'2004*, pages 873–890, Larnaca, Cyprus, October 2004.

[25] Yannis Tzitzikas, Anastasia Analyti, and Nicolas Spyratos. "The Semantics of the Compound Term Composition Algebra". In *Procs. of the 2nd Intern. Conference on Ontologies, Databases and Applications of Semantics, ODBASE'2003*, pages 970–985, Catania, Sicily, Italy, November 2003.

[26] Yannis Tzitzikas, Anastasia Analyti, and Nicolas Spyratos. "Compound Term Composition Algebra: The Semantics". *LNCS Journal on Data Semantics*, 2:58–84, 2005.

[27] Yannis Tzitzikas, Anastasia Analyti, Nicolas Spyratos, and Panos Constantopoulos. "An Algebraic Approach for Specifying Compound Terms in Faceted Taxonomies". In *Information Modelling and Knowledge Bases XV, 13th European-Japanese Conference on Information Modelling and Knowledge Bases, EJC'03*, pages 67–87. IOS Press, 2004.

[28] Yannis Tzitzikas, Raimo Launonen, Mika Hakkarainen, Pekka Kohonen, Tero Leppanen, Esko Simpanen, Hannu Tornroos, Pekka Uusitalo, and Pentti Vanska. "FASTAXON: A system for FAST (and Faceted) TAXONomy design". In *Proceedings of 23th Int. Conf. on Conceptual Modeling, ER'2004*, Shanghai, China, November 2004. (an on-line demo is available at http://fastaxon.erve.vtt.fi/).

[29] Yannis Tzitzikas and Carlo Meghini. "Ostensive Automatic Schema Mapping for Taxonomy-based Peer-to-Peer Systems". In *Seventh International Workshop on Cooperative Information Agents, CIA-2003*, pages 78–92, Helsinki, Finland, August 2003. (Best Paper Award).

[30] Yannis Tzitzikas and Carlo Meghini. "Query Evaluation in Peer-to-Peer Networks of Taxonomy-based Sources". In *Proceedings of 19th Int. Conf. on Cooperative Information Systems, CoopIS'2003*, Catania, Sicily, Italy, November 2003.

[31] Yannis Tzitzikas, Nicolas Spyratos, and Panos Constantopoulos. "Mediators over Taxonomy-based Information Sources". *VLDB Journal*, 2004. (to appear).

[32] B. C. Vickery. "Knowledge Representation: A Brief Review". *Journal of Documentation*, 42(3):145–159, 1986.

[33] J. Ziv and A. Lempel. "A Universal Algorithm for Sequential Data Compression". *IEEE Transactions on Information Theory*, 23(3):337–343, 1977.

[34] J. Ziv and A. Lempel. "Compression of Individual Sequences via Variable-rate Coding". *IEEE Transactions on Information Theory*, 24(5):530–536, 1978.

Appendix A: Proofs

In this Appendix, we give the proofs of the propositions appearing in the paper. We also prove an auxiliary lemma.

Proposition 1 Let PT_k be the number of parse trees returned by $ParseTrees(\{F_1, ..., F_k\})$. It holds that $PT_k \leq (k-1)! * 5^{k-1}$.

Proof: We will compute an upper bound for PT_k. Algorithm $ParseTrees(\{F_1, ..., F_k\})$ extends every parse tree $ptree \in ParseTrees(\{F_1, ..., F_{k-1}\})$ with the facet name F_k. Specifically, it creates two parse trees ($ptree_+$ and $ptree_-$), for each terminal node of $ptree$, and three parse trees ($ptree_+$, $ptree_-$, $ptree_{in}$), for each internal node of $ptree$. The number of terminal nodes of $ptree$ is $k-1$, and the number of internal nodes of $ptree$ is at most $k-2$. Thus, for each $ptree \in ParseTrees(\{F_1, ..., F_{k-1}\})$, Algorithm $ParseTrees(\{F_1, ..., F_k\})$ creates $2(k-1) + 3(k-2) = 5k-8$ parse trees. Thus, we can write $PT_1 = 1$ and $PT_i = PT_{i-1} * (5i-8)$ for each $i > 1$. Now as $5i - 8 < 5(i-1)$, it follows that $PT_k \leq 1 * 5(2-1) * ... * 5(k-1) = (k-1)! * 5^{k-1}$. $\diamond$

Lemma 1 Let S, $S' \in P(\mathcal{T})$. If $minimal(S) \subseteq S'$ and $S' \subseteq S$ then $minimal(S) = minimal(S')$.

Proof: Firstly, we will show that $minimal(S) \subseteq minimal(S')$. Let $s \in minimal(S)$. Then, $s \in S'$. Assume that $s \notin minimal(S')$ then there is s' such that $s' \preceq s$, $s' \neq s$, and $s \in S'$. As $S' \subseteq S$, it follows that $s' \in S$, which is impossible. Therefore, $s \in minimal(S')$.

Now, we will show that $minimal(S') \subseteq minimal(S)$. Let $s \in minimal(S')$. Then, $s \in S'$ and thus, $s \in S$. Assume that $s \notin minimal(S)$ then there is s' such that $s' \preceq s$, $s \neq s'$, and $s \in minimal(S)$. As $minimal(S) \subseteq S'$, it follows that $s \in S'$, which is impossible. Thus, $s \in minimal(S)$. $\diamond$

Proposition 3 Let e and e' be two space-minimal and equivalent (i.e. $e \equiv e'$) expressions. If e has the form $e = \oplus_P(e_1, ..., e_n)$, for $n \geq 3$, and

$$\begin{aligned} e' &= \oplus_{P1}(\oplus_{P2}(e_1, ..., e_{n-1}), e_n) \text{ , or} \\ e' &= \oplus_{P1}(\oplus_{P2}(e_1, ..., e_{l-1}), \oplus_{P3}(e_l, ..., e_n)), \text{ where } 2 < l < n \end{aligned}$$

then it holds $size(e) \leq size(e')$.

Proof: Let $S_i = S_{e_i}$, for $i = 1, ..., n$. It easy to see that it holds:
(1) $\oplus_{P_1}(S_1, ..., S_{n-1}) \oplus_{P_2} S_n = \oplus_{P_1 \cup P_2}(S_1, ..., S_n)$.
(2) $(\oplus_{P_1}(S_1, ..., S_{l-1})) \oplus_{P_2} (\oplus_{P_3}(S_l, ..., S_n)) = \oplus_{P_1 \cup P_2 \cup P_3}(S_1, ..., S_n)$, where $2 < l < n$.
As e is space-minimal, it follows from (1) that $P \subseteq P_1 \cup P_2$. Additionally, it follows from (2) that $P \subseteq P_1 \cup P_2 \cup P_3$. $\diamond$

Proposition 4 Let e and e' be two space-minimal and equivalent (i.e. $e \equiv e'$) expressions. If e has the form $e = \ominus_N(e_1, ..., e_n)$, for $n \geq 3$, and

$$\begin{aligned} e' &= \ominus_{N1}(\ominus_{N2}(e_1, ..., e_{n-1}), e_n) \text{ , or} \\ e' &= \ominus_{N1}(\ominus_{N2}(e_1, ..., e_{l-1}), \ominus_{N3}(e_l, ..., e_n)), \text{ where } 2 < l < n \end{aligned}$$

then it holds $size(e) \leq size(e')$.
Proof: Let $S_i = S_{e_i}$, for $i = 1, ..., n$. It holds:
(1) $\ominus_{N_1}(S_1, ..., S_{n-1}) \ominus_{N_2} S_n = \ominus_{N_1 \cup N_2}(S_1, ..., S_n)$.
(2) $(\ominus_{N_1}(S_1, ..., S_{l-1})) \ominus_{N_2} (\ominus_{N_3}(S_l, ..., S_n)) = \ominus_{N_1 \cup N_2 \cup N_3}(S_1, ..., S_n)$, where $2 < l < n$.
Proof of (1): It holds $\ominus_{N_1}(S_1, ..., S_{n-1}) \ominus_{N_2} S_n = (S_1 \oplus ... \oplus S_{n-1} - Nr(N_1)) \oplus S_n - Nr(N_2)$.
Additionally, it holds $\ominus_{N_1 \cup N_2}(S_1, ..., S_n) = (S_1 \oplus ... \oplus S_n - Nr(N_1)) - Nr(N_2)$.
Therefore, it is enough to show that $(S_1 \oplus ... \oplus S_{n-1} - Nr(N_1)) \oplus S_n = S_1 \oplus ... \oplus S_n - Nr(N_1)$.

We will first show that $(S_1 \oplus ... \oplus S_{n-1} - Nr(N_1)) \oplus S_n \subseteq S_1 \oplus ... \oplus S_n - Nr(N_1)$. Let $s \in (S_1 \oplus ... \oplus S_{n-1} - Nr(N_1)) \oplus S_n$. We will show that $s \notin Nr(N_1)$. If $s \in S_1 \oplus ... \oplus S_{n-1} - Nr(N_1)$ or $s \in S_n$ then obviously $s \notin Nr(N_1)$. Otherwise, $s = s_1 \cup s_2$, where $s_1 \in S_1 \oplus ... \oplus S_{n-1} - Nr(N_1)$ and $s_2 \in S_n$ such that $s_1, s_2 \neq \emptyset$. Assume that $s \in Nr(N_1)$. Then, there is $s' \in N_1$ such that $s \preceq s'$. From the construction of S_i and the fact that $N_1 \subseteq G_{S_1,...,S_{n-1}}$, there are no t, t' such that $t \in s_2$, $t' \in s'$ and $t \leq t'$. From this, it follows that $s_1 \preceq s'$. Therefore, $s_1 \in Nr(N_1)$, which is impossible. Therefore, $s \notin Nr(N_1)$. Thus, $s \in S_1 \oplus ... \oplus S_n - Nr(N_1)$.

Let $s \in S_1 \oplus ... \oplus S_n - Nr(N_1)$. We will show that $s \in (S_1 \oplus ... \oplus S_{n-1} - Nr(N_1)) \oplus S_n$. If $s \in S_1 \oplus ... \oplus S_{n-1}$ then obviously $s \in S_1 \oplus ... \oplus S_{n-1} - Nr(N_1)$. Otherwise, $s = s_1 \cup s_2$, where $s_1 \in S_1 \oplus ... \oplus S_{n-1}$ and $s_2 \in S_n$ such that $s_1, s_2 \neq \emptyset$. If $s_1 \in Nr(N_1)$ then $s \in Nr(N_1)$, which is impossible. Therefore, $s_1 \notin Nr(N_1)$. Thus, $s \in (S_1 \oplus ... \oplus S_{n-1} - Nr(N_1)) \oplus S_n$.

Proof of (2): Similarly to the proof of (1).

As e is space-minimal, it follows from (1) that $N \subseteq N_1 \cup N_2$. Additionally, it follows from (2) that $N \subseteq N_1 \cup N_2 \cup N_3$. $\diamond$

Proposition 5 Let M be a materialized faceted taxonomy, and let e_i, $i = 1, .., n$, be well-formed expressions such that $f(e_i) \cap f(e_j) = \emptyset$, for $i \neq j$. Let $S_{e_i} = V_\circ(M) \cap P(\mathcal{T}_{e_i})$, for $i = 1, ..., n$. It holds:

$$minimal(G_{S_{e_1},...,S_{e_n}} \cap V_\circ(M)) = minimal(G_{S_{e_1},...,S_{e_n}} \cap \pi_e(V_\circ^+(M))),$$
where $e = \oplus_P(e_1, ..., e_n)$ or $e = \ominus_N(e_1, ..., e_n)$

Proof: Recall the following definitions: $V_\circ^+(M) = minimal(V_\circ(M) \cap G_{T_1,...,T_k})$, and
$\pi_e(S) = \{s_{f(e)} \mid s \in S\}$.
Firstly, we will prove that $minimal(G_{S_{e_1},...,S_{e_n}} \cap V_\circ(M)) \subseteq G_{S_{e_1},...,S_{e_n}} \cap \pi_e(V_\circ^+(M))$.
Let $s \in minimal(G_{S_{e_1},...,S_{e_n}} \cap V_\circ(M))$. Then, $s \in G_{S_{e_1},...,S_{e_n}}$ and $s \in \pi_e(V_\circ(M) \cap G_{T_1,...,T_k})$. Assume that $s \notin minimal(\pi_e(V_\circ(M) \cap G_{T_1,...,T_k}))$. Then, there is compound term s' such that $s' \preceq s, s' \neq s$, and $s' \in \pi_e(V_\circ(M) \cap G_{T_1,...,T_k})$. From this, it follows that $s' \in V_\circ(M)$ and $s' \subseteq P(\mathcal{T}_e)$. As $s' \preceq s$, $s \in G_{S_{e_1},...,S_{e_n}}$, and $s' \subseteq P(\mathcal{T}_e) \cap V_\circ(M)$, it follows that $s' \in G_{S_{e_1},...,S_{e_n}}$. Thus, $s' \in G_{S_{e_1},...,S_{e_n}} \cap V_\circ(M)$, which is impossible. Therefore, $s \in minimal(\pi_e(V_\circ(M) \cap G_{T_1,...,T_k})) \subseteq \pi_e(V_\circ^+(M))$. Now, it follows that $s \in G_{S_{e_1},...,S_{e_n}} \cap \pi_e(V_\circ^+(M))$.

As $minimal(G_{S_{e_1},...,S_{e_n}} \cap V_\circ(M)) \subseteq G_{S_{e_1},...,S_{e_n}} \cap \pi_e(V_\circ^+(M))$ and $G_{S_{e_1},...,S_{e_n}} \cap \pi_e(V_\circ^+(M)) \subseteq G_{S_{e_1},...,S_{e_n}} \cap V_\circ(M)$, it follows from Lemma 1 of this Appendix that $minimal(G_{S_{e_1},...,S_{e_n}} \cap V_\circ(M)) = minimal(G_{S_{e_1},...,S_{e_n}} \cap \pi_e(V_\circ^+(M)))$. $\diamond$

Proposition 6 Let M be a materialized faceted taxonomy, and let e_i, $i = 1, .., n$, be well-formed expressions such that $f(e_i) \cap f(e_j) = \emptyset$, for $i \neq j$. Let $S_{e_i} = V_\circ(M) \cap P(\mathcal{T}_{e_i})$, for $i = 1, ..., n$. It holds:

$$maximal(G_{S_{e_1},...,S_{e_n}} - V_\circ(M)) = G_{S_{e_1},...,S_{e_n}} \cap V_\circ^-(M)$$

Proof: Recall the following definition: $V_\circ^-(M) = maximal(G_{T_1,...,T_k} - V_\circ(M))$.

Firstly, we will prove that $maximal(G_{S_{e_1},...,S_{e_n}} - V_\circ(M)) \supseteq G_{S_{e_1},...,S_{e_n}} \cap V_\circ^-(M)$.
Let $s \in G_{S_{e_1},...,S_{e_n}} \cap V_\circ^-(M)$. Then, $s \in G_{S_{e_1},...,S_{e_n}}$ and $s \in maximal(G_{T_1,...,T_k} - V_\circ(M))$. Thus, $s \in G_{S_{e_1},...,S_{e_n}} - V_\circ(M)$. Assume that $s \notin maximal(G_{S_{e_1},...,S_{e_n}} - V_\circ(M))$. Then, there is compound term s' such that $s \preceq s', s' \neq s$, and $s' \in G_{S_{e_1},...,S_{e_n}} - V_\circ(M)$. Thus, $s' \in G_{T_1,...,T_k} - V_\circ(M)$, which is impossible. Therefore, $s \in maximal(G_{S_{e_1},...,S_{e_n}} - V_\circ(M))$.

Now, we will prove that $maximal(G_{S_{e_1},...,S_{e_n}} - V_\circ(M)) \subseteq G_{S_{e_1},...,S_{e_n}} \cap V_\circ^-(M)$.
Let $s \in maximal(G_{S_{e_1},...,S_{e_n}} - V_\circ(M))$. Then, $s \in G_{S_{e_1},...,S_{e_n}}$ and $s \in G_{T_1,...,T_k} - V_\circ(M)$. Assume

that $s \notin maximal(G_{T_1,...,T_k} - V_\circ(M))$. Then, there is compound term s' such that $s \preceq s', s' \neq s$, and $s' \in G_{T_1,...,T_k} - V_\circ(M)$. As $s \in G_{S_{e_1},...,S_{e_n}}$ and $s \preceq s'$, it follows that $s' \in S_{e_1} \oplus ... \oplus S_{e_n}$. As $s' \in G_{T_1,...,T_k} - V_\circ(M)$, it follows that $s' \notin V_\circ(M)$ and thus, $s' \notin S_{e_1} \cup ... \cup S_{e_n}$. Therefore, $s' \in G_{S_{e_1},...,S_{e_n}} - V_\circ(M)$, which is impossible. Thus, $s \in maximal(G_{T_1,...,T_k} - V_\circ(M)) = V_\circ^-(M)$. Now, it follows that $s \in G_{S_{e_1},...,S_{e_n}} \cap V_\circ^-(M)$. $\diamond$

Appendix B: Table of Symbols

Symbol	Definition
$P(.)$	$Powerset$
$s \preceq s'$	$\forall t' \in s' \ \exists t \in s$ such that $t \leq t'$
T_i	$\{\{t\} \mid t \in \mathcal{T}_i\} \cup \{\emptyset\}$
$S_1 \oplus ... \oplus S_n$	$\{ s_1 \cup ... \cup s_n \mid s_i \in S_i\}$
$\oplus_P(S_1, ...S_n)$	$S_1 \cup ... \cup S_n \ \cup \ Br(P)$
$\ominus_N(S_1, ...S_n)$	$S_1 \oplus ... \oplus S_n - Nr(N)$
$\overset{*}{\oplus}(T_i)$	$P(\mathcal{T}_i)$
$\overset{*}{\oplus}_P(T_i)$	$T_i \cup Br(P)$
$\overset{*}{\ominus}_N(T_i)$	$\overset{*}{\oplus}(T_i) - Nr(N)$
$G_{S_1,...,S_n}$	$S_1 \oplus ... \oplus S_n - \cup_{i=1}^n S_i$
G_{T_i}	$\overset{*}{\oplus}(T_i) - T_i$
S_e	the evaluation of an expression e
$\bar{I}(t)$	$\cup\{I(t') \mid t' \leq t\}$
$\hat{\bar{I}}(\{t_1, ..., t_n\})$	$\bar{I}(t_1) \cap ... \cap \bar{I}(t_n)$
$I^{-1}(o)$	$\{t \in \mathcal{T} \mid o \in I(t)\}$
$V(M)$	$\{s \in P(\mathcal{T}) \mid \hat{\bar{I}}(s) \neq \emptyset\}$
$V_\circ(M)$	$V(M) \cap (T_1 \oplus ... \oplus T_k)$
$V_\circ^+(M)$	$minimal(V_\circ(M) \cap G_{T_1,...,T_k})$
$V_\circ^-(M)$	$maximal(G_{T_1,...,T_k} - V_\circ(M))$
$f(t)$	the facet of term t
$f(s)$	$\{f(t) \mid t \in s\}$
$f(e)$	the facets that appear in expression e
$s_{f(e)}$	$\{t \in s \mid f(t) \in f(e)\}$
$\pi_i(s)$	$\{t \in s \mid f(t) = F_i\}$
$\pi_e(S)$	$\{s_{f(e)} \mid s \in S\}$

Table 6: Table of Symbols