Mining the Meaningful Compound Terms from Materialized Faceted Taxonomies

Yannis Tzitzikas[1] Anastasia Analyti[2]

[1] *Institut d'Informatique, F.U.N.D.P. (University of Namur), Belgium*
[2] *Institute of Computer Science, ICS-FORTH, Greece*
Email : ytz@info.fundp.ac.be, analyti@csi.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 that have non-empty interpretation. The mined expressions can be used for encoding compactly (and subsequently reusing) the domain knowledge that is stored in existing materialized faceted taxonomies. Furthermore, expression mining is very crucial for reorganizing taxonomy-based sources which were not initially designed according to a clear faceted approach (like the directories of Google and Yahoo!), so as to have a semantically clear, and compact faceted structure. We analyze this problem and we give an analytical description of all algorithms needed for expression mining.
Keywords : Materialized faceted taxonomies, knowledge extraction and reuse, algorithms

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 (or *invalid*), in the sense they cannot be applied to any object of the domain. For instance, we cannot do any winter sport in the

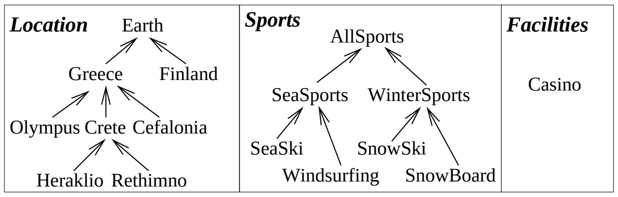


Fig. 1. A faceted taxonomy for indexing hotel Web pages

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, suppose that only in Cefalonia there exists a hotel that has a casino, and that this hotel also offers sea ski and wind-surfing 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 2 (found in Appendix A).

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 [19]. 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) [19] (which is recalled in Section 2.2) can significantly reduce the required effort. According to that approach the designer can use an algebraic expression to define the valid 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 2, can be defined using the expression:

$$e = (\mathit{Location} \ominus_N \mathit{Sports}) \oplus_P \mathit{Facilities}$$

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) 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 the 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 2). 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.

Moreover, it can be exploited for reorganizing single-hierarchical (non-faceted) materialized taxonomies (like the directories of Yahoo! or Google), so as to give them a clear faceted structure but *without* loosing the knowledge encoded in their taxonomy. 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 [19]), which unlike the existing navigational trees of the single-hierarchical taxonomies, do not present the problem of missing terms or missing relationships (for more about this problem see [5]). 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 partition their terminologies to a set of facets, using languages like the one proposed in [17] (we will not elaborate this problem in this paper), and then use the algorithms presented in this paper for expression mining. One important remark that we have to mention here is that the closed world assumptions of the algebraic operations of the CTCA (described in [18]) lead us to a remarkably high "compression ratio". 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 [20], which are needed in order to build mediators or peer-to-peer systems over this kind of sources [21]. Figure 2 illustrates graphically our problem and its context. Other applications of expression mining are presented in the concluding section.

Note that here "mining" does not have the classical statistical nature, i.e. we are not trying to mine statistically justified associations or rules [7]. Instead, from an object base we are trying to mine the associations of the elements of

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

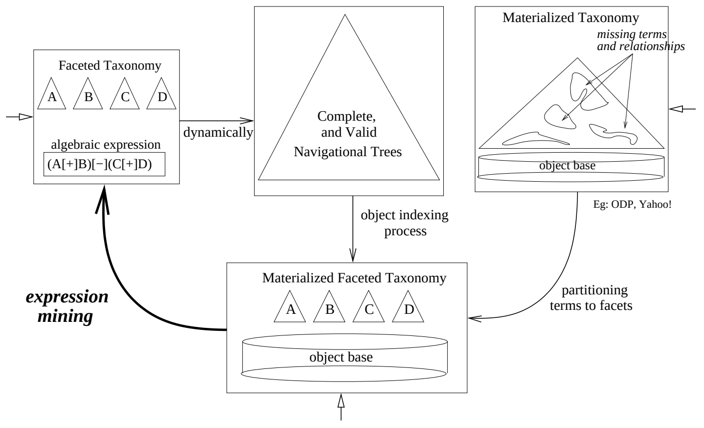


Fig. 2. The application context of expression mining

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*.

The rest of this paper is organized as follows: Section 2 describes the required background and Section 3 states the problem. Section 4 describes straightforward methods for extracting an algebraic expression that specifies the valid compound terms of a materialized faceted taxonomy. Section 5 describes the method and the algorithms for finding the *shortest*, i.e. most compact and efficient expression. Additionally, it gives a demonstrating example. Finally, Section 6 concludes the paper and identifies issues for further research. A table of symbols is given in Appendix B.

2 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 [19, 18]. In subsection 2.3, we define materialized faceted taxonomies.

2.1 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 $\forall t' \in s'\ \exists t \in s$ 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'. Figure 3(a) shows the compound ordering over the compound terms of our previous example. Note that $s_1 \preceq s_3$, as s_1 contains $SeaSki$ which is a term narrower than the unique term $Sports$ of s_3. On the other hand, $s_1 \not\preceq s_2$, as s_1 does not contain a term narrower than $WinterSports$. Finally, $s_2 \preceq s_3$ and $s_3 \preceq \emptyset$. In fact, $s \preceq \emptyset$, for every compound term s.

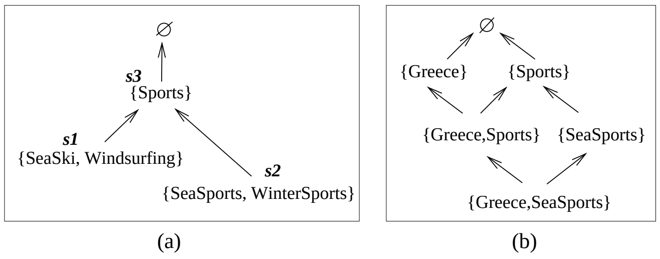


Fig. 3. Two examples of compound taxonomies

A *compound taxonomy* over $\mathcal{T}$ is a pair $(S, \preceq)$, where S is a compound terminology over $\mathcal{T}$, 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: $\text{Br}(s) = \{s' \in P(\mathcal{T}) \mid s \preceq s'\}$ and $\text{Nr}(s) = \{s' \in P(\mathcal{T}) \mid s' \preceq s\}$. The broader and the narrower compound terms of a compound terminology S are defined as follows: $Br(S) = \cup\{\text{Br}(s) \mid s \in S\}$ and $Nr(S) = \cup\{\text{Nr}(s) \mid s \in S\}$.

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 $\mathcal{T}$. This compound taxonomy is shown in Figure 3.(b). For reasons of brevity, hereafter we shall omit the term $\emptyset$ from the compound terminologies of our examples and figures.

2.2 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 [19, 18]).

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 1.

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

Table 1. 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 4, and consists of 8 terms.

Now consider the compound terminologies S and S' shown in the left part of Figure 5, 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.

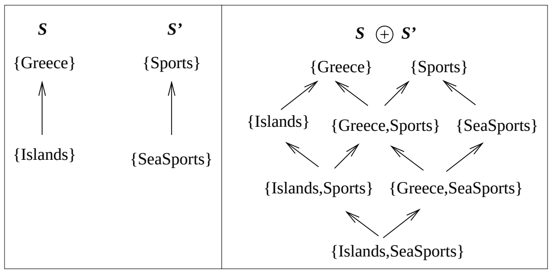


Fig. 4. An example of a product $\oplus$ operation

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 5. 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 1).

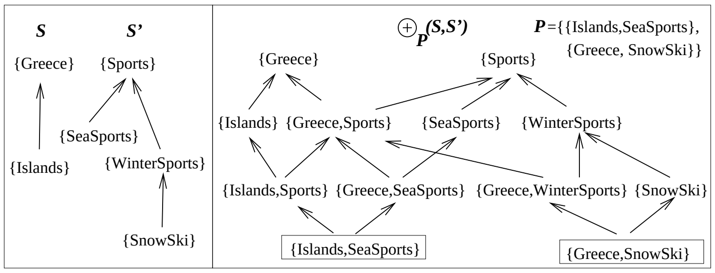


Fig. 5. An example of a *plus-product*, $\oplus_P$, operation

Alternatively, we can obtain the compound taxonomy shown at the right part of Figure 5 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 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. According to our semantics, all compound terms in S_e are *valid*, and the rest in $P(\mathcal{T}) - S_e$ are *invalid* [18].

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 terminologies. 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 $e = \oplus_P(e_1, e_2)$ then it should be $P \subseteq G_{S_{e_1},S_{e_2}}$, and
(iii) each parameter N that appears in e, is a subset of the associated set of genuine compound terms, e.g. if $e = \overset{*}{\ominus}_N (T_i)$ then it should be $N \subseteq G_{T_i}$.

For example, the expression[6] $(T_1 \oplus_P T_2) \ominus_N T_1$ is not well-formed, as T_1 appears twice in the expression. Constraints (i), (ii), and (iii) ensure that the evaluation of an expression is monotonic, meaning that the valid and invalid compound terms of an expression e increase as the length of e increases. For example, if we omit constraint (i) then an invalid compound term according to an expression $T_1 \oplus_P T_2$ could be valid according to a larger expression $(T_1 \oplus_P T_2) \oplus_{P'} T_1$. If we omit constraint (ii) then an invalid compound term according to an expression $T_1 \oplus_{P_1} T_2$ could be valid according to a larger expression $(T_1 \oplus_{P_1} T_2) \oplus_{P_2} T_3$. Additionally, if we omit constraint (iii) then a valid compound term according to an expression $T_1 \oplus_P T_2$ could be invalid according to a larger expression $(T_1 \oplus_P T_2) \ominus_N T_3$.

In the rest of the paper, *we consider only well-formed expressions.* In [19], we presented the algorithm $IsValid(e, s)$ that takes as input a (well-formed) expression e and a compound term s, and checks whether $s \in S_e$. This algorithm has polynomial time complexity, specifically $O(|\mathcal{T}|^2 * |s| * |\mathcal{P} \cup \mathcal{N}|)$, where $\mathcal{P}$ denotes the union of all P parameters of e, and $\mathcal{N}$ denotes the union of all N parameters of e.

Additionally, [18] defines the semantics of CTCA and shows why we cannot use Description Logics [8] to represent 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

[6] For binary operations, we also use the infix notation.

implemented by VTT and Helsinki University of Technology (HUT) under the name FASTAXON [3]. The system is currently under experimental evaluation.

2.3 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 6, where the objects are denoted by natural numbers. This will be the running example of our paper.

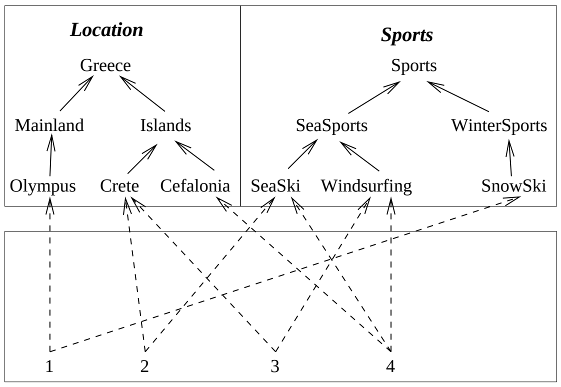


Fig. 6. A materialized faceted taxonomy

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: $\hat{I}(t) = I(t)$, $\hat{I}(q \wedge q') = \hat{I}(q) \cap \hat{I}(q')$, $\hat{I}(q \vee q') = \hat{I}(q) \cup \hat{I}(q')$, $\hat{I}(q \wedge \neg q') = \hat{I}(q) \setminus \hat{I}(q')$. One can easily see that a compound term $\{t_1, ..., t_k\}$ actually corresponds to a conjunction $t_1 \wedge ... \wedge t_k$. 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 $\hat{\bar{I}}(\{SeaSki, Windsurfing\}) = \{4\}$.

3 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]

The following table indicates the valid compound terms of the materialized faceted taxonomy shown in Figure 6, that contain exactly one term from each facet.

	Greece	Mainl.	Olymp.	Islands	Crete	Cefal.
Sports	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$
SeaSports	$\surd$			$\surd$	$\surd$	$\surd$
SeaSki	$\surd$			$\surd$	$\surd$	$\surd$
Windsurf.	$\surd$			$\surd$	$\surd$	$\surd$
WinterSp.	$\surd$	$\surd$	$\surd$			
SnowSki	$\surd$	$\surd$	$\surd$			

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

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

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 $S_e = V(M)$, 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 [19], 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 $S_e = V(M)$.*

One important remark is that solving the above problem allow us to solve also the following:

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

[8] Recall that the time complexity of Alg. $IsValid(e, s)$ [19] is proportional to $size(e) = |\mathcal{P} \cup \mathcal{N}|$.

Problem 3: *Given an expression e', 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 $V(M)$ 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.

For simplicity, in this paper we do not consider self-product operations. Their inclusion is a trivial extension of the presented methods. Therefore, from $V(M)$ we consider only the compound terms that contain at most one term from each facet.

4 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 basic compound terminologies, i.e. are not singletons. One can easily see that $S_{\oplus_{P'}(T_1,...,T_k)} = V(M)$. 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 $S_{\oplus_P(T_1,...,T_k)} = V(M)$. By applying the above two-step algorithm to our current example we get that:

$$\begin{aligned} P = \{&\{Olympus, SnowSki\}, \{Crete, SeaSki\}, \\ &\{Crete, Windsurfing\}, \{Cefalonia, SeaSki\}, \\ &\{Cefalonia, Windsurfing\}\} \end{aligned}$$

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

The first step computes all invalid compound terms that contain at most one term from each facet. One can easily see that $S_{\ominus_{N'}(T_1,...,T_k)} = V(M)$. 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 $S_{\ominus_N(T_1,...,T_k)} = V(M)$. 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 that of the plus-product expression (the parameter N contains only 2 compound terms, while P contains 5).

5 Mining the Shortest Expression

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

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 $FindShortestExpression(\ \mathcal{F}, V)$ (Alg. 51) which takes as input a faceted taxonomy $\mathcal{F}$ and a set of compound terms V, and returns the *shortest* expression e over $\mathcal{F}$ such that $S_e = V$. 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, the procedure $ParseTrees(\{F_1, ..., F_n\})$ (which is described in detail in subsection 5.1) takes as input a set of facets $\{F_1, ..., F_n\}$ and returns all possible parse trees of the expressions over $\{F_1, ..., F_n\}$.

Now the procedure $SpecifyParams(e, V)$ (which is described in detail in subsection 5.2) takes as input a parse tree e and a set of compound terms $V \subseteq P(\mathcal{T})$, and *specifies* the parameters P and N of e such that $S_e = V$.

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

[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.

Finally, the algorithm $FindShortestExpression(\mathcal{F},V)$ returns the shortest expression e such that $S_e = V$. Summarizing, $FindShortestExpression(\mathcal{F},V(M))$ returns the solution to Problem 2.

Algorithm 51 $FindShortestExpression(\mathcal{F}, V)$
Input: A faceted taxonomy $\mathcal{F}$ generated by $\{F_1, ..., F_k\}$, and a set of compound terms V
Output: The shorthest expression e such that $S_e = V$

```
$minSize$ := MAXINT;
$shortestExpr$ := "";
For each $e$ in **ParseTrees**($\{F_1, ..., F_k\}$) do
    $e'$ := **SpecifyParams**($e$, $V$);
    $size$ := **GetSize**($e'$);
    If $size < minSize$ then
        $minSize$ := $size$;
        $shortestExpr$ := $e'$;
    EndIf
EndFor
return ($shortestExpr$)
```

5.1 Deriving all Possible Parse Trees

In this subsection, we describe how we can compute the parse trees of all possible expressions over a set of facets $\{F_1, ..., F_n\}$. Recall that the parse tree of an expression is a tree structure that describes a derivation of the expression according to the rules of the grammar. A depth-first-search traversal of the parse tree of an expression e can be used to obtain the prefix form of the expression e. In our case, the *terminal* (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. For example, Figure 7(c) displays all different parse trees for the set of facets $\{A, B, C\}$. Note that every facet appears just once in a parse tree, as we consider only well-formed expressions.

Algorithm $ParseTrees(\{F_1, ..., F_n\})$ (Alg. 52) takes as input a set of facet names $\{F_1, ..., F_n\}$ and returns all possible parse trees for $\{F_1, ..., F_n\}$. We will first exemplify the algorithm and its reasoning through a small example.

Consider the facets $\{A, B, C\}$ of our current example. We will use a recursive method for computing the parse trees for $\{A, B, C\}$. At first, we find the parse trees for $\{A\}$. Clearly, there is only one parse tree for $\{A\}$, and it consists of a single node with name A (see Figure 7(a)). Subsequently, we find the parse trees

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

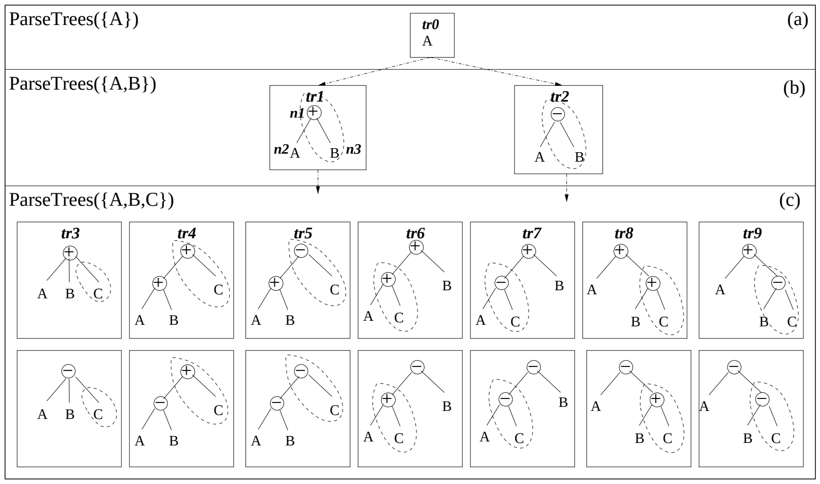


Fig. 7. All possible parse trees for $\{A\}$, $\{A, B\}$, and $\{A, B, C\}$

of $\{A, B\}$. There are two ways for extending the parse tree for $\{A\}$ with the new facet B: (i) by creating a "+" node with children A and B, and (ii) by creating a "-" node with children A and B. Thus, we can create two parse trees for $\{A, B\}$, named tr_1 and tr_2 (see Figure 7(b)). In other words, $ParseTrees(\{A, B\}) = \{tr_1, tr_2\}$, where the parse tree tr_1 corresponds to $\oplus(A, B)$, and the parse tree tr_2 corresponds to $\ominus(A, B)$.

Now, we can find the parse trees of $\{A, B, C\}$, by extending each node of each parse tree in $ParseTrees(\{A, B\})$ with the new facet C. For doing so, initially we visit the parse tree tr_1. At first we visit the internal node n_1 of tr_1 and we extend it in three different ways (all other nodes of tr_1 remain the same):

1. by adding C to the children of n_1. Now n_1 corresponds to the operation $\oplus(A, B, C)$, and this extension results to the parse tree tr_3.
2. by creating a new "+" node with children the nodes n_1 and C. The new node corresponds to $\oplus(\oplus(A, B), C)$, and this extension results to the parse tree tr_4.
3. by creating a new "-" node with children the nodes n_1 and C. The new node corresponds to $\ominus(\oplus(A, B), C)$, and this extension results to the parse tree tr_5.

Now, we visit the terminal node $n2$ of tr_1 and we extend it in two different ways:

1. by creating a new "+" node with children the nodes n_2 and C. The new node corresponds to the operation $\oplus(A, C)$, and this extension results to the parse tree tr_6.
2. by creating a new "-" node with children the nodes n_2 and C. The new node corresponds to the operation $\ominus(A, C)$, and this extension results to the parse tree tr_7.

Finally, we visit the terminal node $n3$ of tr_1 and we extend it in two different ways, similarly to node n_2. These extensions result to the parse trees tr_8 and tr_9.

After finishing with tr_1, we visit tr_2 and we extend each node of tr_2 with the new facet C, similarly to tr_1. Figure 7(c) gives all the parse trees for $\{A, B, C\}$. Generally, the above process is repeated recursively until all the facets of a faceted taxonomy have been considered.

Algorithm 52 $ParseTrees(\{F_1, ..., F_n\})$
Input: A set $\{F_1, ..., F_n\}$ of facet names
Ouptut: All possible parse trees for $\{F_1, ..., F_n\}$

```
(1) If $n = 1$ then return($\{$CreateNode$(F_1)\}$);
(2) $allPtrees := \{\}$;
(3) For each $ptree \in$ ParseTrees$(\{F_1, ..., F_{n-1}\})$ do
(4)     $allPtrees := allPtrees \cup$ ExtendedTrees$(ptree, ptree, F_n)$;
(5) return($allPtrees$)
```

Below we will describe in detail the algorithms needed for deriving all possible parse trees. Given a node n we shall use n.`Parent` to refer to the parent of n, and `Children`(n) to the children of node n. We shall also use the following auxiliary routines: `CreateNode`(nm) a function that creates and returns a new node with name nm, and `IsTerminal`(n) a function that returns true if n is a terminal node, and false otherwise.

Let us now describe in detail the algorithm $ParseTrees(\ \{F_1, ..., F_n\})$ (Alg. 52). The procedure $ParseTrees(\{F_1, ..., F_n\})$ calls $ParseTrees(\{F_1, ..., F_{n-1}\})$. Then, for each parse tree *ptree* returned by $ParseTrees(\{F_1, ..., F_{n-1}\})$, it issues the call $ExtendedTrees$(*ptree*, *ptree*, F_n).

Let us now see what $ExtendedTrees(ptree, extNode, Fn)$ (Alg. 53) does. The procedure takes as input a parse tree *ptree*, a node $extNode$[11] of the *ptree*, and a facet name Fn. It returns a *set* of parse trees that correspond to the extension of *ptree* with the new facet name Fn, at the node $extNode$. Now the way the extension is performed depends on the kind of the node $extNode$ (i.e. terminal or internal). Specifically, there are two cases:

C1: $extNode$ is a terminal node (say F_i).
In this case the lines (3)-(6) produce two copies of the *ptree* (called $ptree_+$ and $ptree_-$, respectively), and call the routine $ExtendTreeNode$ that does that actual extension. After the execution of these lines,
$ptree_+$ corresponds to the extension $\oplus(F_i, F_n)$, and
$ptree_-$ corresponds to the extension $\ominus(F_i, F_n)$.
The exact algorithm for $ExtendTreeNode$ is presented below in this section (Alg. 54). The function $TreeCopy$(*ptree*) takes as input a parse tree *ptree*

[11] It is named $extNode$, because the operation corresponding to that node will be extended with the new facet name F_n.

and returns a copy, say *ptree_copy*, of *ptree*. Notice that according to line (1), *ptree* keeps a pointer $ExtNode$ to the node $extNode$. After the call of $TreeCopy$(*ptree*), *ptree_copy*.$ExtNode$ points to the copy of the $extNode$ in the *ptree_copy*.

C2: $extNode$ is an internal node (i.e. either "+" or "-").
This means that $extNode$ corresponds to either a $\oplus(e_1, ..., e_i)$ or a $\ominus(e_1, ..., e_i)$ operation. Below we shall write $\odot(e_1, ..., e_i)$ to denote any of the above two operations.
In this case the routine *ExtendTreeNode* is called three times (lines (10)-(14)): These calls produce three copies of *ptree*, namely $ptree_{in}$, $ptree_{+}$, and $ptree_{-}$, where:
$ptree_{in}$ corresponds to the extension $\odot(e_1, ..., e_i, T_n)$,
$ptree_{+}$ corresponds to the extension $\oplus(\odot(e_1, ..., e_i), T_n)$, and
$ptree_{-}$ corresponds to the extension $\ominus(\odot(e_1, ..., e_i), T_n)$.
At last, the routine $ExtendedTrees(ptree, extNode, Fn)$ calls itself ($ExtendedTrees(ptree,\ childNode,\ Fn)$), for each child $childNode$ of the node $extNode$.

Algorithm 53 $ExtendedTrees(ptree, extNode, Fn)$
Input: a parse tree *ptree*, a node $extNode$ of *ptree*, and a facet name Fn
Output: a set of parse trees that correspond to the extension of *ptree* with the new facet name Fn, at the node $extNode$

```
(1)  $ptree.ExtNode := extNode$;
(2)  If **IsTerminal**($extNode$) then
(3)      $ptree_{+}$ := **TreeCopy**($ptree$);
(4)      **ExtendTreeNode**($ptree_{+}.ExtNode$, "+", $Fn$);
(5)      $ptree_{-}$ := **TreeCopy**($ptree$);
(6)      **ExtendTreeNode**($ptree_{-}.ExtNode$, "-", $Fn$);
(7)      return($\{ptree_{+}, ptree_{-}\}$)
(8)  Else           // extNode is an internal node
(9)      $ptree_{+}$ := **TreeCopy**($ptree$);
(10)     **ExtendTreeNode**($ptree_{+}.ExtNode$, "+", $Fn$);
(11)     $ptree_{-}$ := **TreeCopy**($ptree$);
(12)     **ExtendTreeNode**($ptree_{-}.ExtNode$, "-", $Fn$);
(13)     $ptree_{in}$ := **TreeCopy**($ptree$);
(14)     **ExtendTreeNode**($ptree_{in}.ExtNode$, "in", $Fn$);
(15)     $extendedTrees$ :=$\{ptree_{in}, ptree_{+}, ptree_{-}\}$;
(16)     For each $childNode \in$ **Children**($extNode$) do
(17)         $extendedTrees := extendedTrees \cup$
                 **ExtendedTrees**($ptree, childNode, Fn$) ;
(18)     return($extendedTrees$)
(19) End if
```

Algorithm 54 $ExtendTreeNode(extNode, flag, Fn)$
Input: a node $extNode$ of a parse tree, a flag $flag$ that denotes the type of the extension with the new facet name Fn, and a facet name Fn
Output: the parse tree extended with Fn at the $extNode$, according to $flag$

```
(1) $FnNode$ :=CreateNode($Fn$) ;

(2) If $flag$ =”in” then
(3)     $FnNode$.Parent := $extNode$ ;

(4) If $flag$=”+” or $flag$=”-” then
(5)     $newOpNode$ :=CreateNode($flag$);
(6)     $FnNode$.Parent := $newOpNode$ ;
(7)     InsertBetween($extNode$.Parent, $extNode$, $newOpNode$);
(8) End if
```

Notice that Alg. 54 uses the function `InsertBetween`$(nUp, nDown, new)$. This function inserts the node new between the nodes nUp and $nDown$. This means that after this call, it holds $nDown$.`Parent` = new and new.`Parent` = nUp. Clearly, if nUp is nil then new becomes root node.

As an example, Figure 7(b) shows the output of $ParseTrees(\{A, B\})$. Figure 7(c) shows the output of $ParseTrees(\{A, B, C\})$. The first row of the parse trees in Figure 7(c) corresponds to the parse trees returned by $ExtendedTrees(tr1, tr1, C)$ and the second row corresponds to the parse trees returned by $ExtendedTrees(tr2, tr2, C)$.

As a final comment note that some parse trees returned by $ParseTrees(\{F_1, ..., F_k\})$ can been eliminated before $SpecifyParams(e, V)$ is called by algorithm $FindShortestExpression(\mathcal{F}, V)$ (Alg. 51), for setting up the parameters. This is because, these parse trees cannot yield to the shortest expression. For example, consider the parse trees $tr3$ and $tr4$ of Figure 7(c). The parse tree $tr4$ can be eliminated, because $tr4$ can never yield to an expression with size less than that of $tr3$. Actually, 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 for yielding the shortest expression.

5.2 Specifying the Parameters

This section describes the algorithm $SpecifyParams(e, V)$ (Alg. 55), i.e. an algorithm that takes as input the parse tree of an expression e (with undefined P and N parameters) and a set of compound terms $V \subseteq P(\mathcal{T})$, and returns the same parse tree that is now enriched with P and N parameters that satisfy the condition $S_e = V$. Of course, this is possible only if $Br(V) = V$ (note that $Br(V(M)) = V(M)$).

Algorithm 55 $SpecifyParams(e, V)$
Input: The parse tree of an expression e, and a set of compound terms $V \subseteq P(\mathcal{T})$
Output: The parse tree of e enriched with P and N parameters, such that $S_e = V$

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

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 $SpecifyParams(e_i, V)$ is called recursively, for all $i = 1, ..., n$. Subsequently, the statement $P' := G_{S_{e_1},...,S_{e_n}} \cap V$ computes and stores in P' those elements of V 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 4).

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 $SpecifyParams(e_i, V)$ is called recursively, for all $i = 1, ..., n$. Subsequently, the statement $N' := G_{S_{e_1},...,S_{e_n}} \setminus V$ computes and stores in N' those elements of $G_{S_{e_1},...,S_{e_n}}$ that are invalid, i.e. not in V (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 4).

For example, consider the four-faceted taxonomy $\mathcal{F}$ shown in Figure 8(a), and suppose that V is the set of compound terms shown in Figure 8(b). Below we give the trace of execution of $SpecifyParams(e, V)$ for the expression $e = \oplus_P(A, \ominus_N(B, C, D))$:

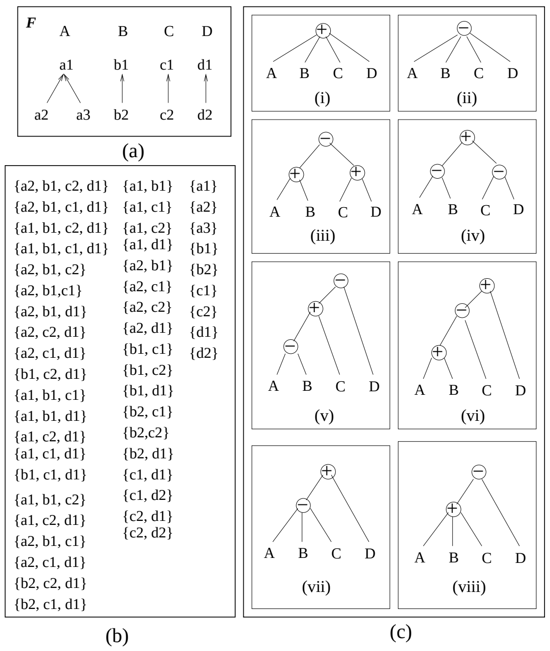
```
call $SpecifyParams(\oplus_P(A, \ominus_N(B, C, D)), V)$
    call $SpecifyParams(\ominus_N(B, C, D), V)$
        return // $N$ is set equal to $\{\{b2, d2\}\}$
    return // $P$ is set equal to $\{\{a2, b1, c2, d1\}\}$
```

So the parameters of the resulting expression e are the following:

$$P = \{\{a2, b1, c2, d1\}\}$$
$$N = \{\{b2, d2\}\}$$

5.3 An Example of Shortest Expression Mining

Let us now apply the above algorithms to the four-faceted taxonomy shown in Figure 8(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 $V(M)$ consists of the 48 compound terms listed in Figure 8(b). For simplification, in that figure we do not show the interpretation I, but directly the set $V(M)$.



{a2, b1, c2, d1}	{a1, b1}	{a1}
{a2, b1, c1, d1}	{a1, c1}	{a2}
{a1, b1, c2, d1}	{a1, c2}	{a3}
{a1, b1, c1, d1}	{a1, d1}	{b1}
{a2, b1, c2}	{a2, b1}	{b2}
{a2, b1,c1}	{a2, c1}	{c1}
{a2, b1, d1}	{a2, c2}	{c2}
{a2, c2, d1}	{a2, d1}	{d1}
{a2, c1, d1}	{b1, c1}	{d2}
{b1, c2, d1}	{b1, c2}	
{a1, b1, c1}	{b1, d1}	
{a1, b1, d1}	{b2, c1}	
{a1, c2, d1}	{b2,c2}	
{a1, c1, d1}	{b2, d1}	
{b1, c1, d1}	{c1, d1}	
{a1, b1, c2}	{c1, d2}	
{a1, c2, d1}	{c2, d1}	
{a2, b1, c1}	{c2, d2}	
{a2, c1, d1}		
{b2, c2, d1}		
{b2, c1, d1}		

Fig. 8. An example of expression mining

The algorithm $FindShortestExpression(\mathcal{F}, V(M))$ 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 8(c) sketches some indicative parse trees). Then, for each parse tree e in the output, it calls the procedure *SpecifyParams*$(e, V(M))$, which assigns particular values to the parameters P and N of e such that $S_e = V(M)$. The sizes of all derived expressions are compared to get the shortest expression, which is the following:

$$e = \oplus_P(A, \ominus_N(B, C, D)), \text{ where}$$

$$P = \{\{a2, b1, c2, d1\}\}$$
$$N = \{\{b2, d2\}\}$$

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.

6 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.

Specifically, we confine ourselves to the case of materialized faceted taxonomies (for more about faceted classification see [16, 9, 22, 11, 15]). Materialized faceted taxonomies are employed in several different domains, including Libraries [12], Software Repositories [13, 14], Web catalogs and many others. Current interest in faceted taxonomies is also indicated by several ongoing projects like FATKS[12], FACET[13], FLAMENGO[14], and the emergence of XFML [1] (Core-eXchangeable Faceted Metadata Language) that aims at applying the faceted classification paradigm on the Web.

In this paper we showed how we can find 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. The size of the resulting expressions is remarkably low. In particular, we gave two straightforward methods for extracting a plus-product and a minus-product expression (possibly, none the shortest), and an exhaustive algorithm for finding the shortest expression. The complexity of the latter is of course 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.

[12] http://www.ucl.ac.uk/fatks/database.htm
[13] http://www.glam.ac.uk/soc/research/hypermedia/facet_proj/index.php
[14] http://bailando.sims.berkeley.edu/flamenco.html

Expression mining can be exploited for encoding compactly the set of valid compound terms of materialized faceted taxonomies. This can significantly aid their exchange and reuse. It also worths mentioning here that the recently emerged XFML+CAMEL [2] (*C*ompound term composition *A*lgebraically-*M*otivated *E*xpression *L*anguage) allows publishing and exchanging faceted taxonomies *and* CTCA expressions using an XML format.

Expression mining also allows the reorganization of single-hierarchical materialized taxonomies (such as Yahoo! and ODP), so as to give them a faceted structure without missing the knowledge encoded in their pre-coordinated terms. Such a reorganization would certainly facilitate their management, extension, and reuse. Furthermore, it would allow the dynamic derivation of "complete" and valid navigational trees for this kind of sources [19].

Expression mining can also be exploited in *language engineering*. WordNet [6] is a lexical ontology where 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. 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 [10]). Notice that in this scenario, the number of valid combinations of words would be so big, that storing them explicitly would be quite problematic. So, this scenario also indicates the need for expression mining, in order to extract and represent compactly the valid combinations of words.

Another application of expression is *query answering optimization*. Suppose a materialized faceted taxonomy that stores a very large number of objects at which users can pose conjunctive queries (i.e. compound terms) in order to find the objects of interest. The availability of the set of valid compound terms would allow realizing efficiently whether a query has an empty or a non-empty answer. In this way, we can avoid performing wasteful computations for queries whose answer will be eventually empty.

Issues for further research include measuring the exact time/space complexity of expression mining, designing optimal algorithms for shortest expression mining, and investigating whether it worths employing additional indexing data structures (e.g. labelling schemes like those mentioned in [4]) for further speedup.

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. "FASTAXON: A system for FAST (and Faceted) TAXONomy design.", 2004. To be published as an open source project in http://opensource.erve.vtt.fi/fastaxon.
4. Vassilis Christophides, Dimitris Plexousakis, Michel Scholl, and Sotirios Tourtounis. On Labeling Schemes for the Semantic Web. In *Proceedings of WWW'2003*, pages 544–555, Budapest, Hungary, May 2003.
5. Peter Clark, John Thompson, Heather Holmback, and Lisbeth Duncan. "Exploiting a Thesaurus-based Semantic Net for Knowledge-based Search". In *Proceedings of 12th Conf. on Innovative Applications of AI (AAAI/IAAI'00)*, pages 988–995, 2000.
6. Princeton University Cognitive Science Laboratory. "WordNet: A Lexical Database for the English Language". (http://www.cogsci.princeton.edu/ wn).
7. Padhraic Smyth David J. Hand, Heikki Mannila. *Principles of Data Mining.* MIT Press, 2001. ISBN: 1-57735-027-8.
8. 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.
9. Elizabeth B. Duncan. "A Faceted Approach to Hypertext". In Ray McAleese, editor, *HYPERTEXT: theory into practice, BSP*, pages 157–163, 1989.
10. 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.
11. P. H. Lindsay and D. A. Norman. *Human Information Processing.* Academic press, New York, 1977.
12. Amanda Maple. "Faceted Access: A Review of the Literature", 1995. http://theme.music.indiana.edu/tech_s/mla/facacc.rev.
13. Ruben Prieto-Diaz. "Classification of Reusable Modules". In *Software Reusability. Volume I*, chapter 4, pages 99–123. acm press, 1989.
14. Ruben Prieto-Diaz. "Implementing Faceted Classification for Software Reuse". *Communications of the ACM*, 34(5):88–97, 1991.
15. 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.
16. 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.
17. 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.
18. Yannis Tzitzikas, Anastasia Analyti, and Nicolas Spyratos. "The Semantics of the Compound Terms 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.
19. 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.
20. 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).

21. 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.
22. B. C. Vickery. "Knowledge Representation: A Brief Review". *Journal of Documentation*, 42(3):145–159, 1986.

Appendix A: Example

In this Appendix, we present the valid and invalid compound terms of the example of Figure 1. For simplicity, we consider only the compound terms that contain at most one term from each facet.

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

Invalid	
Crete, WinterSp.	Cefalonia, WinterSp.
Rethimno, WinterSp.	Heraklio, WinterSp.
Olympus, SeaSki	Olympus, WindSurf.
Crete, SnowBoard	Cefalonia, SnowBoard
Rethimno, SnowBoard	Heraklio, SnowBoard
Crete, SnowSki	Cefalonia, SnowSki
Rethimno, SnowSki	Heraklio, SnowSki
Finland, Cas.	Olympus, Cas.
Crete, Cas.	Heraklio, Cas.
Rethimno, Cas.	WinterSp., Cas.
SnowBoard, Cas.	SnowSki, Cas.
Olympus, SeaSports	Crete, WinterSp., Cas.
Cefalonia, WinterSp., Cas.	Rethimno, WinterSp., Cas.
Heraklio, WinterSp., Cas.	Olympus, SeaSki, Cas.
Olympus, WindSurf., Cas.	Crete, SnowBoard, Cas.
Cefalonia, SnowBoard, Cas.	Rethimno, SnowBoard, Cas.
Heraklio, SnowBoard, Cas.	Crete, SnowSki, Cas.
Cefalonia, SnowSki, Cas.	Rethimno, SnowSki, Cas.
Heraklio, SnowSki, Cas.	Olympus, AllSports, Cas.
Crete, AllSports, Cas.	Rethimno, AllSports, Cas.
Heraklio, AllSports, Cas.	Crete, SeaSports, Cas.
Rethimno, SeaSports, Cas.	Heraklio, SeaSports, Cas.
Olympus, WinterSp., Cas.	Crete, SeaSki, Cas.
Rethimno, SeaSki, Cas.	Heraklio, SeaSki, Cas.
Crete, WindSurf., Cas.	Rethimno, WindSurf., Cas.
Heraklio, WindSurf., Cas.	Olympus, SnowBoard, Cas.
Olympus, SnowSki, Cas.	Finland, AllSports, Cas.
Finland, SeaSports, Cas.	Finland, WinterSp., Cas.
Finland, SeaSki, Cas.	Finland, WindSurf., 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 2. The Valid and Invalid compound terms of the example of Figure 1

As the facet *Location* has 8 terms, the facet *Sports* has 7 terms, and the facet *Facilities* has one term, the number of compound terms that contain at most 1 term from each facet is 9*8*2 = 144. This table contains 60 valid and 67 invalid compound terms, thus 127 in total. By adding the (8+7+1=16) singletons (which were omitted from the column of valid) and the empty set we reach the 144.

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
$\bar{I}(t)$	$\cup\{I(t') \mid t' \leq t\}$
$\hat{\bar{I}}(\{t_1, ..., t_n\})$	$\bar{I}(t_1) \cap ... \cap \bar{I}(t_n)$
$V(M)$	$\{s \in P(\mathcal{T}) \mid \hat{\bar{I}}(s) \neq \emptyset\}$

Table 3. Table of Symbols