On-demand Generation of AOC-posets: Reducing the Complexity of Conceptual Navigation

Alexandre Bazin, Jessie Carbonnel, Giacomo Kahn

On-demand Generation of AOC-posets: Reducing the Complexity of Conceptual Navigation

Alexandre Bazin[1], Jessie Carbonnel[2], and Giacomo Kahn[1]

[1] LIMOS & Université Clermont Auvergne, France
giacomo.kahn@isima.fr, contact@alexandrebazin.com
[2] LIRMM, CNRS & Université de Montpellier, France
jessie.carbonnel@lirmm.fr

Abstract. Exploratory search allows to progressively discover a dataspace by browsing through a structured collection of documents. Concept lattices are graph structures which support exploratory search by conceptual navigation, i.e., navigating from concept to concept by selecting and deselecting descriptors. These methods are known to be limited by the size of concept lattices which can be too large to be efficiently computed or too complex to be browsed intelligibly. In this paper, we address the problem of providing techniques that reduce the complexity of FCA-based exploratory search. We show the suitability of AOC-posets, a condensed alternative structure to achieve conceptual navigation. Also, we outline algorithms to enable an on-demand generation of AOC-posets. The necessity to devise more flexible methods to perform product selection in software product line engineering is what motivates our work.

Keywords: Formal Concept Analysis, AOC-poset, Concept Navigation, Software Product Line Engineering, Product Selection.

1 Introduction

Exploratory search is an information retrieval strategy that aims at guiding the user into a space of existing documents to help him select the one that best suits his needs. This process is particularly adapted to situations where a user is unfamiliar with the dataspace, or when the data is too large to be known entirely. Lattice structures were among the first structures used to support information retrieval processes [9], and their usage was later generalised to Formal Concept Analysis (FCA) theory [8]. The concept lattice offers a convenient structure to do exploratory search, where navigating from concept to concept by selecting or deselecting attributes emulates iterative modifications of the document descriptor selection, and thus of the current research state. Exploratory search by conceptual navigation has been used in several applications, for instance querying web documents [4] or browsing a collection of images [6]. However, FCA-based exploratory search raises some problems, mainly because of the size (in terms of number of concepts) of lattices, which are well known to grow exponentially with

the size of the input data. Computing the whole concept lattice can take time and it needs adapted algorithms to be efficiently used in applications. Moreover, a user can rapidly get disoriented while navigating in such a large and convoluted structure. Therefore, several ways to overcome these limitations have been studied in the literature [11, 13, 7].

In this paper, we propose a new and more scalable approach to perform exploratory search by conceptual navigation, that relies on local generation of AOC-posets, a partial sub-order of concept lattices. Unlike concept lattices, which depict all possible queries a user can formulate, this alternative conceptual structure represents and structures the minimal set of queries that are necessary to perform conceptual navigation, and therefore permit navigation through a less complex structure. Also, to avoid generating the whole AOC-poset, we only generate the current concept and its neighbourhood, represented by its direct sub-concepts and super-concepts. In fact, even though an AOC-poset is smaller than its associated concept lattice, it still can be advantageous to only generate the parts of the structure we are interested in, especially in large datasets. We outline algorithms to identify neighbour concepts in AOC-posets, i.e., determining upper and lower covers of a given concept in an AOC-poset. An application of exploratory search in the field of software product line engineering, for an activity called product selection, is what motivates our work.

The remainder of this paper is organised as follows. In Sect. 2, we present our motivations in the domain of software product line engineering. In Sect. 3, we study AOC-posets to perform exploratory search. We then propose algorithms to compute upper and lower covers of a concept in AOC-posets in Sect. 4, and we test our approach on existing datasets. Related work is discussed in Sect. 5, and Sect. 6 concludes and presents some future work.

2 Motivation

Software product line engineering (SPLE) [14] is a development paradigm that aims to efficiently create and manage a collection of related software systems. SPLE is based on factorisation and exploitation of a common set of artifacts, organised around a generic architecture from which several software variants can be derived. A central point of SPLE is the modelisation of the common parts and the variants contained in the related software systems, called the *variability* of the software product line (SPL). This variability is represented by variability models, which are the traditional starting points to perform information retrieval operations on SPLs, including product selection, an important task that consists in guiding the user into selecting the functionalities he wants in the final derived software system. The most prevalent approach to model variability relies on *feature models* (FMs) [12], a family of visual languages that describe a set of features (i.e., main characteristics) and dependencies between them. Figure 1 depicts an FM representing an SPL about cell phones. A combination of features respecting all the constraints expressed in the FM is called a *valid configuration*, and corresponds to a derivable software system.

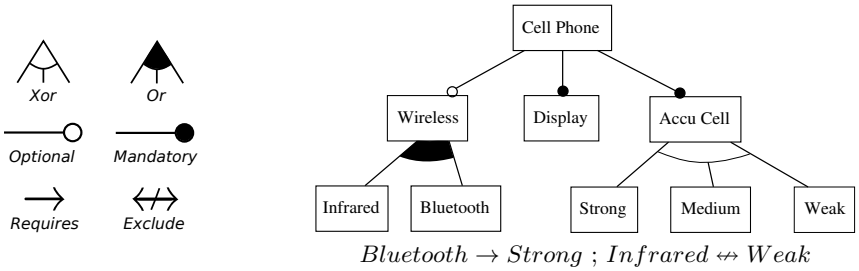

Fig. 1. Excerpt of a feature model representing an SPL about Cell Phones

Current approaches for product selection rely on the feature model hierarchy to automatically deploy configurators; however, these methods are too stiff considering that it does not allow the user to change his final configuration without having to start again the product selection, or to see which other configurations are similar to his. We propose to apply exploratory search in the context of product selection to complement these methods and offer a more flexible selection. In fact, conceptual navigation allows a user to start from an existing or partial configuration, explore similar ones, and be informed on how he can select or deselect features to obtain these configurations. It is noteworthy that the number of valid configurations depicted by an FM grows exponentially with its number of features. To be able to conceive applications using conceptual navigation in this context, reducing the complexity of the underlying conceptual structure along with its generation time is crucial.

3 AOC-poset: a Condensed Structure for Conceptual Navigation

Formal Concept Analysis (FCA) [8] is a mathematical framework that structures a set of objects described by attributes depending on the attributes they share. As input, FCA takes a *formal context* $K = (\mathcal{O}, \mathcal{A}, \mathcal{R})$, where $\mathcal{O}$ is the set of objects, $\mathcal{A}$ the set of attributes and $\mathcal{R} \subseteq \mathcal{O} \times \mathcal{A}$ a binary relation. A pair (a, o) from $\mathcal{R}$ states that "the object o possesses the attribute a". Table 1 presents the formal context representing the 7 valid configurations of the FM of Fig. 1.

The application of FCA permits to extract from a context K a finite set of *formal concepts* through the use of two *derivation operators* $(\cdot)'$; $(\cdot)' : 2^{\mathcal{O}} \mapsto 2^{\mathcal{A}}$, and $(\cdot)' : 2^{\mathcal{A}} \mapsto 2^{\mathcal{O}}$. Thus, $O' = \{a \in \mathcal{A} \mid \forall o \in O, (o, a) \in \mathcal{R}\}$ and $A' = \{o \in \mathcal{O} \mid \forall a \in A, (o, a) \in \mathcal{R}\}$. A formal concept C is a pair (E, I) with $E \subseteq \mathcal{O}$ and $I \subseteq \mathcal{A}$, representing a maximal set of objects that share a maximal set of common attributes. $E = I'$ is the concept's *extent* (denoted $Ext(C)$), and $I = E'$ is the concept's *intent* (denoted $Int(C)$). The set of all concepts extracted from K together with the extent set-inclusion order forms a lattice structure called a *concept lattice*. Figure 2 (left) presents the concept lattice associated with the formal context of Table 1. We simplify the representation of intents and

Table 1. Formal context depicting the 7 configurations of the SPL about cell phones

	Cell Phone	**Wireless**	**Infrared**	**Bluetooth**	**Display**	**Accu Cell**	**Strong**	**Medium**	**Weak**
c_1	x				x	x	x		
c_2	x				x	x		x	
c_3	x				x	x			x
c_4	x	x	x		x	x	x		
c_5	x	x	x		x	x		x	
c_6	x	x		x	x	x	x		
c_7	x	x	x	x	x	x	x		

extents in the lattice by displaying each attribute (resp. object) only once in the structure, in the lowest (resp. the greatest) concept having this attribute (resp. object). We say that these concepts *introduce* an element. The attributes of a concept are inherited from top to bottom, and the objects from bottom to top.

A concept introducing at least an attribute is called an *attribute-concept* (AC), and a concept introducing at least an object is called an *object-concept* (OC). A concept can introduce both an attribute and an object (*attribute-object-concept* (AOC)), or it can introduce neither of them (*plain-concept*). Plain-concepts appear in the lattice as concepts with empty extents and intents. In some types of applications, it is not necessary to take these concepts into account. For instance, this is the case when the lattice is only used as a support to organise objects and their attributes (therefore represented by their introducer concepts), and not to highlight maximal groups of elements. In these particular cases, one can benefit from only generating the sub-order restricted to the *introducer* concepts instead of the whole concept lattice. This smaller structure (in terms of number of concepts) is called an *Attribute-Object-Concept partially ordered set* (AOC-poset) [10]. Figure 2 (right) presents the AOC-poset associated with the context from Table 1: it corresponds to the partial order of concepts from Fig. 2 (left), minus Concept_0 and Concept_7. While a concept lattice can have up to $2^{min(|\mathcal{A}|,|\mathcal{O}|)}$ concepts, the associated AOC-poset cannot exceed $|\mathcal{O}| + |\mathcal{A}|$ concepts.

AOC-posets can be used as a smaller alternative to concept lattices to 1) structure a collection of objects depending on the attributes they share and 2) navigate through this collection by selecting and deselecting attributes. But, if concept lattices represent all possible queries a user can formulate, AOC-posets restrict this set to the minimal queries required to perform conceptual navigation. As we have seen before, neighbour concepts in a concept lattice represent minimal possible modifications a user can make to the current query and therefore offer a dataspace in which one can navigate in minimal steps. This means that concept lattices allow to select and deselect non-cooccurrent attributes one by one. AOC-posets do not preserve the minimal step query refinement/enlargement property, but factorise the possible query modification steps to keep the most prevalent ones. For instance, in the concept lattice of Fig. 2 (left), if a user has selected Concept_4 as the current concept, he can choose to deselect attribute *Bluetooth*

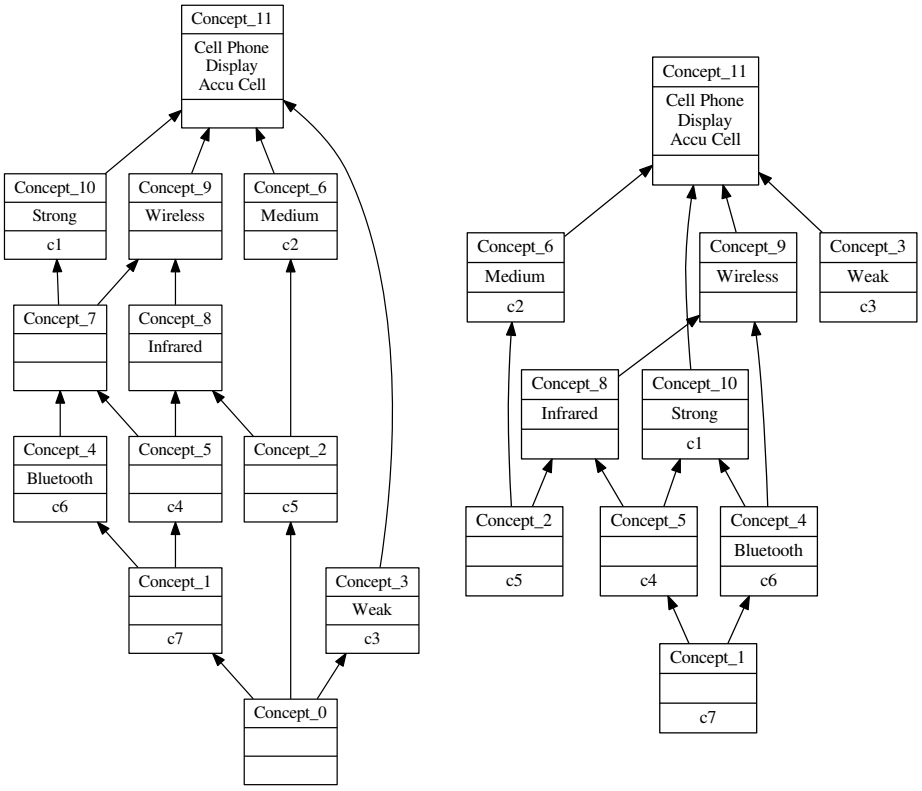

Fig. 2. Concept lattice (left) and AOC-poset (right) associated with the formal context of Table 1

and thus move to Concept_7. From this concept, he can now choose to deselect either *Strong* or *Wireless* and move respectively to Concept_9 or Concept_10. In AOC-posets, because plain-concepts, playing the role of "transition steps", are not present, selection/deselection choices to move from concept to concept are condensed. This time, in Fig. 2 (right), if a user want to enlarge its query from Concept_4, he can either deselect both *Bluetooth* and *Strong* in one step to move to Concept_9, or deselect both *Bluetooth* and *Wireless* to reach Concept_10.

4 Partial Generation of AOC-poset

On-demand, or local, generation consists in generating only the part of the structure we are interested in, and has already been applied to concept lattices, with algorithms such as nextClosure [8]. To our knowledge, several algorithms exist to build AOC-posets: Ares, Ceres, Pluton and Hermes [3]. However, none of them perform on-demand generation of AOC-posets. In what follows, we outline algorithms to retrieve the neighbourhood of a given concept in an AOC-poset.

4.1 Computing Upper and Lower Covers of a Concept in the AOC-poset

Exploration can start from the top concept (i.e., the most general query), in the case where the user wants to make a software configuration from scratch. But, it is possible that the user already has partial knowledge of the configuration he wants, and it is then necessary to be able to start from any concept in the AOC-poset. As the concept corresponding to the (potentially partial) configuration that the user has in mind does not necessarily introduce an object or an attribute, we suppose the input of the exploration is a formal concept, plain or not. The problem is thus to compute the upper and lower covers of a given concept C_i in the AOC-poset.

Let us start with computing the upper covers. We are looking for the smallest ACs or OCs greater than the input. We start out by computing the smallest ACs greater than C_i. They can be obtained by computing the concepts $(\{a\}', \{a\}'')$ for each attribute $a \in Int(C_i)$. We remark that $(\{a_1\}', \{a_1\}'') \geq (\{a_2\}', \{a_2\}'')$ if and only if $a_1 \in \{a_2\}''$. As such, the smallest ACs are the ones that are computed from attributes that do not appear in the closures of other attributes. Once we have the smallest ACs, we want to compute the smallest OCs that are between them and C_i. This means that we are looking for concepts $(\{o\}'', \{o\}')$ such that o is in the extent of one of the ACs we have and $\{o\}' \subset Int(C_i)$. We remark once again that $(\{o_1\}', \{o_1\}'') \geq (\{o_2\}', \{o_2\}'')$ implies $o_2 \in \{o_1\}''$ and that the closures of some objects give us information on OCs that can't be minimal.

Algorithm 1 computes the upper neighbours of the input concept in the AOC-poset. The first loop computes the closure of single attributes. Each closure allows us to remove attributes that correspond to non-minimal ACs. The resulting set R contains the intents of the ACs that are both super-concepts of C_i and minimal for this property. The second loop constructs the set O of objects that are in the extent of an element of R but not in the extent of C_i. The third loop removes the objects of O that cannot possibly be introduced by a superset of C_i and, finally, the fourth loop removes the objects of O that produce non-minimal OCs. The ACs that are no longer minimal are also removed. Therefore, considering the inital configuration, the OCs introduce the most similar and more generalised configurations, and the ACs show the factorised possible attribute de-selections the user can make.

Computing the lower covers is done using the same algorithm, exchanging the roles of attributes and objects. This time, OCs present the most similar and more specialised configurations, and the ACs the possible condensed attribute selections.

4.2 Implementation

We have implemented our algorithms, and we tested them on SPL datasets extracted from the `SPLOT` repository[3]. `SPLOT` (for *Software Product Line Online*

[3] http://www.splot-research.org/

Algorithm 1: Upper Covers

Input: A concept C_i
Output: The upper covers of C_i in the AOC-poset

$A \leftarrow Int(C_i)$
foreach $a \in A$ **do**
  $Y \leftarrow \{a\}''$
  $A \leftarrow A \setminus \{Y \setminus \{a\}\}$
$R \leftarrow \{\{a\}''|a \in A\}$
$O \leftarrow \emptyset$
forall $S \in R$ **do**
  $X \leftarrow S'$
  $O \leftarrow O \cup (X \setminus Ext(C_i))$
forall $o \in O$ **do**
  if $o' \not\subset Int(C_i)$ **then**
    $O \leftarrow O \setminus \{o\}$
forall $o \in O$ **do**
  $T = \{S|(S \in R) \wedge (o \in S')\}$
  $R \leftarrow R \setminus T$
  $Y \leftarrow \{o\}''$
  if $\exists p \in O$ *such that* $p \in Y$ **then**
    $O \leftarrow O \setminus \{o\}$
$R \leftarrow \{(\{o\}'', \{o\}') \mid o \in O\}$
return R

Tools) is an academic website providing a repository of feature models along with a set of tools to create, edit and perform automated analysis on them. We have selected 13 representative feature models which describe SPLs as e-shops, cell phones or video games, from small sizes (13 configurations) to larger ones (4774 configurations). To test our method on data extracted from feature models, we first create a formal context $configurations \times features$ for each one of them. Then, from a context, our implementation permits to find a concept corresponding to a subset of features and compute its conceptual neighbourhood in AOC-posets. In this experiment, we assume that a user will not exceed 50 navigation steps, as he wants to be familiarised to the similar valid configurations around his initial selection of features. To measure the gain of our method in terms of number of computed concepts, we compare for each context the number of computed concepts for 50 navigation steps (i.e., a concept and its neighbourhood) to the total number of concepts in the associated AOC-poset and concept lattice. The results are presented in Fig. 3. The concept lattice curve permits to indicate an "upper-bound" to visualise more easily the gain of AOC-posets and local-generation of AOC-posets comparing to concept lattices.

Figure 3 shows that both aspects of our method are useful for conceptual structures with a size around 700+ concepts. AOC-posets are smaller than con-

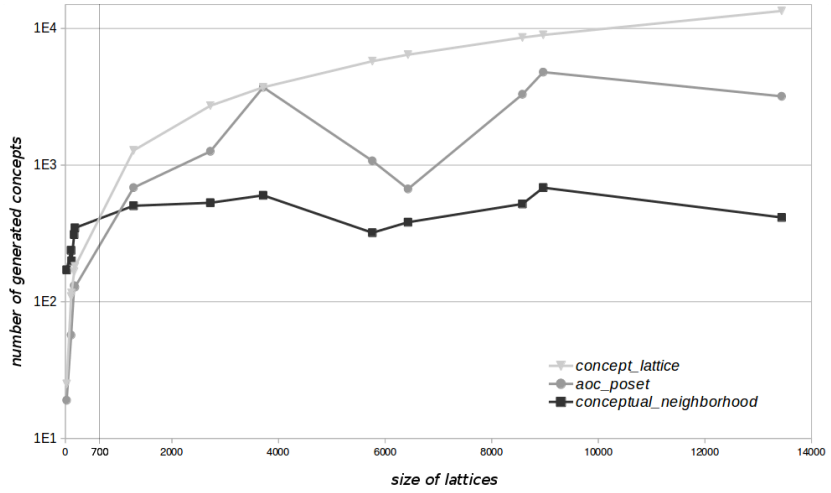

Fig. 3. Number of generated concepts for AOC-posets and conceptual neighbourhood for 50 navigation steps, depending on the size of their associated concept lattices (logarithmic scale)

cept lattices, but the gap between the two conceptual structures can grow exponentially with their sizes. In fact, the difference when the structures are small is not very important (e.g., 19 concepts against 25, 131 against 166), but AOC-posets can become very interesting with larger structures (e.g., 1074 concepts against 5761, 669 against 6430). Performing several navigation steps in a small structure makes re-computation of same concepts more likely. In these cases, it is preferable to compute the whole AOC-poset from the begining. Our experiment shows that this is the case when the initial conceptual structure possesses less than around 700 concepts, and with 50 navigation steps.

5 Related Work

Several methods have been proposed through the literature to reduce the complexity of conceptual navigation. In [9], the authors choose not to show the whole concept lattice to the user, but only a part of it, restricted to a focus concept and its neighbourhood. This navigation approach, which we study and apply in this paper, can be found in several works [6, 5, 1]. In [13], the authors propose two methods to extract trees from concept lattices and use them as less complex structures for browsing and visualisation. The difference between the two methods lies in the way the "best" parent for each concept in the tree is assigned: the first one is based on the selection of one parent per layer, and the second one on conceptual indexes. They then simplify again the final structure by applying two reduction methods based on fault-tolerance and clustering on the extracted trees. In [2], the authors propose a tool to build and visualise formal concept trees.

Carpineto and Romano [4] allowed the user to bound the information space by dynamically applying constraints during the search to prune the concept lattice. Bounding allows to reduce the explorable dataspace and help the user focus on the parts he is interested in. Following the same idea, iceberg concept lattices [15] are pruned structures that only show the top-part of concept lattices which can be used to perform conceptual navigation. By comparison, we use a partial sub-order of concept lattices. In [5], the authors present `SearchSleuth`, a tool for local analysis of web queries based on FCA, that derives a concept and its neighbourhood from a query. Because the domain cannot be computed entirely, it generates a new formal context at each navigation step: for each user query, it retrieves the list of results, extracts the relevant terms, and builds a context from these terms and their associated documents. The navigation is managed through an interface which suggests terms, making implicit the underlying graph structure and its complexity. Alam et al. [1] present a tool, `LatViz`, that provides several operations to reduce the information space. One of them facilitates the visualisation and the navigation. The authors propose to display the concept lattice *level-wise*: selecting a concept at a level n displays all its sub-concept at level n-1. Another functionality allows to prune the concept lattice by restricting navigation in sub-concepts and/or super-concepts of concepts selected by the user, in the same way as in [4]. Also, the tool permits to compute AOC-posets to support conceptual navigation: the authors describe AOC-posets as the "core" of their corresponding concept lattices. However, they compute the whole structure using the Hermes algorithm and do not propose an on-demand generation. Greene el al. [11] discuss refinement and enlargement (broadening) approaches which are not restricted to neighbour concepts, and therefore allow navigation by non-minimal steps to ease exploratory search in large information spaces.

6 Conclusion and Future Work

In this paper, we adress the problem of providing scalable and praticable techniques to perform conceptual navigation with formal concept analysis. Product selection, a software product line engineering task that can benefit from exploratory search by conceptual navigation to complement current methods which lack of flexibility is the motivation of our work. We used AOC-posets, the concept lattice sub-hierarchy restricted to introducer concepts, as a smaller, condensed alternative to concept lattices that preserve objects and attributes taxonomy. We show that AOC-posets depict the minimal set of queries necessary to browse the dataspace, as they "factorise" the possible attribute selection and deselection steps. To avoid generating the whole sub-hierarchy, we outline algorithms to enable on-demand generation of AOC-posets by computing the neighbourhood of any concept in the AOC-poset. We implemented these algorithms to test our approach on a dozen of SPLs extracted from the `SPLOT` repository. These experiments reveal that our method provides a gain in terms of number of generated concepts when used instead of concept lattices, when the concept lattice has a size larger than about 700 concepts to perform 50 navigation steps.

In the future, we are considering further experiments on different datasets to provide a more complete evaluation of the gain of the proposed approach. From a more theoretical point of view, we plan to extend exploratory search by conceptual navigation to relational data using Relational Concept Analysis.

References

1. Alam, M., Le, T.N.N., Napoli, A.: LatViz: A New Practical Tool for Performing Interactive Exploration over Concept Lattices. In: 13th International Conference on Concept Lattices and Their Applications (CLA). pp. 9–20 (2016)
2. Andrews, S., Hirsch, L.: A tool for creating and visualising formal concept trees. In: 5th Conceptual Structures Tools & Interoperability Workshop (CSTIW 2016) held at the 22nd Int. Conf. on Conceptual Structures (ICCS 2016). pp. 1–9 (2016)
3. Berry, A., Huchard, M., McConnell, R.M., Sigayret, A., Spinrad, J.P.: Efficiently Computing a Linear Extension of the Sub-hierarchy of a Concept Lattice. In: 3rd Int. Conf. on Formal Concept Analysis (ICFCA). pp. 208–222 (2005)
4. Carpineto, C., Romano, G.: Exploiting the Potential of Concept Lattices for Information Retrieval with CREDO. Journal of Universal Computer Science 10(8), 985–1013 (2004)
5. Ducrou, J., Eklund, P.W.: SearchSleuth: The Conceptual Neighbourhood of an Web Query. In: 5th International Conference on Concept Lattices and Their Applications (CLA) (2007)
6. Ducrou, J., Vormbrock, B., Eklund, P.W.: FCA-Based Browsing and Searching of a Collection of Images. In: 14th International Conference on Conceptual Structures (ICCS), Conceptual Structures: Inspiration and Application. pp. 203–214 (2006)
7. Ferré, S.: Efficient Browsing and Update of Complex Data Based on the Decomposition of Contexts. In: 17th Int. Conf. on Conceptual Structures (ICCS). pp. 159–172 (2009)
8. Ganter, B., Wille, R.: Formal concept analysis - mathematical foundations. Springer (1999)
9. Godin, R., Gecsei, J., Pichet, C.: Design of a Browsing Interface for Information Retrieval. In: 12th International Conference on Research and Development in Information Retrieval (SIGIR). pp. 32–39 (1989)
10. Godin, R., Mili, H.: Building and Maintaining Analysis-Level Class Hierarchies Using Galois Lattices. In: 8th Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA). pp. 394–410 (1993)
11. Greene, G.J., Fischer, B.: Single-Focus Broadening Navigation in Concept Lattices. In: 3rd Workshop on Concept Discovery in Unstructured Data, co-located with the 13th International Conference on Concept Lattices and Their Applications (CLA). pp. 32–43 (2016)
12. Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E., Peterson, A.S.: Feature-Oriented Domain Analysis (FODA) Feasibility Study. Citeseer (1990)
13. Melo, C.A., Grand, B.L., Aufaure, M.: Browsing Large Concept Lattices through Tree Extraction and Reduction Methods. International Journal of Intelligent Information Technologies (IJIIT) 9(4), 16–34 (2013)
14. Pohl, K., Böckle, G., van der Linden, F.: Software Product Line Engineering - Foundations, Principles, and Techniques. Springer (2005)
15. Stumme, G., Taouil, R., Bastide, Y., Pasquier, N., Lakhal, L.: Computing iceberg concept lattices with Titanic. Data Knowledge Engineering (DKE) 42(2), 189–222 (2002)