Conceptual Navigation for Polyadic Formal Concept Analysis

Sebastian Rudolph[1], Christian Săcărea[2], and Diana Troancă[2](✉)

[1] Technische Universität Dresden, Dresden, Germany
sebastian.rudolph@tu-dresden.de
[2] Babeş-Bolyai, Cluj-Napoca, Romania
{csacarea,dianat}@cs.ubbcluj.ro

Abstract. Formal Concept Analysis (FCA) is a mathematically inspired field of knowledge representation with wide applications in knowledge discovery and decision support. Polyadic FCA is a generalization of classical FCA that instead of a binary uses an arbitrary, n-ary incidence relation to define *formal concepts*, i.e., data clusters in which all elements are interrelated. We discuss a paradigm for navigating the space of such (formal) concepts, based on so-called membership constraints. We present an implementation for the cases $n \in \{2, 3, 4\}$ using an encoding into answer-set programming (ASP) allowing us to exploit highly efficient strategies offered by optimized ASP solvers. For the case $n = 3$, we compare this implementation to a second strategy that uses exhaustive search in the concept set, which is precomputed by an existing tool. We evaluate the implementation strategies in terms of performance. Finally, we discuss the limitations of each approach and the possibility of generalizations to n-ary datasets.

1 Introduction

Conceptual knowledge is closely related to a deeper understanding of existing facts and relationships, but also to the process of arguing and communicating why something happens in a particular way. Formal Concept Analysis (FCA) [6] is a mathematical theory introduced by Wille, being the core of Conceptual Knowledge Processing [24]. It emerged from applied mathematics and quickly developed into a powerful framework for knowledge representation. It is based on a set-theoretical semantics and provides a rich amount of mathematical instruments for the representation, acquisition, retrieval, discovery and further processing of knowledge.

FCA defines concepts as maximal clusters of data in which all elements are mutually interrelated. In FCA, data is represented in a basic data structure, called *formal context*. A *dyadic* formal context consists of two sets, one of *objects* and another of *attributes* and a binary relation between them, expressing which objects have which attributes. From such dyadic formal contexts, *formal concepts* can be extracted using concept forming operators, obtaining a mathematical structure called *concept lattice*. Thereby, the entire information contained in the formal context is preserved. The concept lattice and its graphical representation as an order diagram can then serve as the basis for communication and further data analysis. Navigation in concept lattices enables exploring, searching, recognizing, identifying, analyzing, and investigating; this exemplifies the fruitfulness of this approach for knowledge management.

In subsequent work, Lehmann and Wille extended dyadic FCA to a triadic setting (3FCA) [16], where *objects* are related to *attributes* under certain *conditions*. The *triadic concepts* (short: *triconcepts*) arising from such data, can be arranged in mathematical structures called *trilattices*. Trilattices can be, up to some conditions, graphically represented as a triangular diagram, yet, this kind of knowledge representation is much less useful and intuitive than its dyadic counterpart, because of the difficulties of reading and navigating in such triadic diagrams. Even if the theoretical foundations of trilattices and that of 3FCA have been intensely studied, there is still a need for a valuable navigation paradigm in triadic concept sets. To overcome these difficulties, we proposed in 2015 a navigation method for triadic conceptual landscapes based on a neighborhood notion arising from dyadic concept lattices obtained by projecting along a dimension [20]. This method enables exploration and navigation in triconcept sets by locally displaying a smaller part of the space of triconcepts, instead of displaying all of them at once.

Voutsadakis [22] further generalized the idea from dyadic and triadic to n-adic data sets, introducing the term *Polyadic Concept Analysis*. He describes concept forming operators in the n-dimensional setting as well as the basic theorem of polyadic concept analysis, a generalization of earlier results by Wille [23].

FCA was successfully used on triadic or tetradic ($n = 4$) datasets such as folksonomies [14], data logs of rental services [2] or data about mobile operators [12]. However, a common problem for n-ary concept sets is their size and complexity. Even for $n = 2$ and for relatively small data sets, the number of formal concepts tends to be quite large (it can be of exponential size in the worst case), which makes the graphical representation of these sets in their entirety unusable for practical purposes. Several strategies have been proposed to overcome this problem. For instance, Dragoş et al. [4,5] are using a circular representation for triadic data while investigating users' behavioral patterns in e-learning environments. Săcărea [21] uses a graph theoretical approach to represent triadic concept sets obtained from medical data. For n-adic concept sets with $n \geq 4$, no navigation strategies have been presented yet.

In 2015, we introduced membership constraints for n-adic concepts in order to narrow down the set of user-relevant n-concepts and to focus on a certain data subset one is interested to explore or start exploration from [19]. As opposed to classical navigation tools, conceptual navigation has at its core a formal concept, i.e. a complete cluster of knowledge. We discussed the problem of satisfiability of membership constraints, determining if a formal concept exists whose object and attribute sets include certain elements and exclude others.

In the current paper, we consider a general navigation paradigm for the space of polyadic concepts and implement this paradigm for the dyadic, triadic and tetradic case. For the triadic case, we try two different implementations. The first one uses the capabilities of Answer Set Programming (ASP) for computing concepts and solving the corresponding membership constraint satisfaction problem. By using this strategy, the implementation also explores optimization strategies offered by ASP. The second strategy is based on an exhaustive search of the set of polyadic concepts. The concept set is no longer computed using the ASP encoding but by one of the existing 3FCA tools. Finally, we evaluate the performance of these two strategies in terms of implementation and computation speed and we discuss the limitations of each approach and show that the ASP approach can be extended to any n-ary dataset.

2 Preliminaries

2.1 Formal Concept Analysis

In this section, we briefly present the necessary basic notions and definitions. For a more detailed introduction, please refer to [6].

A dyadic context is defined to be a triple (G, M, I), where G and M are sets and $I \subseteq G \times M$ is a binary relation. The set G is called *set of objects*, M is the *set of attributes* and I is the *incidence relation*. Concepts are extracted herefrom using *derivation operators*. For $A \subseteq G$ we define $A' = \{m \in M \mid \forall g \in A, gIm\}$, i.e. A' consists of all common attributes of all object from A. Dually, for a set $B \subseteq M$ of attributes, we define B' to be the set of objects sharing all attributes from B. The derivation operators are a Galois connection on the power sets of G and M, respectively. A *formal concept* is a pair (A, B) of objects and attributes, respectively, with $A' = B, B' = A$. The set of all formal concepts is ordered by the subconcept-superconcept relationship: If $(A, B), (C, D)$ are concepts, then $(A, B) \leq (C, D)$ if and only if $A \subseteq C$ (which is equivalent to the condition $D \subseteq B$). By this, the set of all concepts becomes a complete lattice, called *concept lattice*.

The triadic case has been studied by Lehmann and Wille ([16]). The fundamental structures used by triadic FCA are those of a triadic formal context and a triadic formal concept, also referred to as triadic concept or short triconcept. For a better understanding, we define the triadic context first, and then explain the notion of formal concept for the general case.

Definition 1. *A triadic formal context $\mathbb{K} = (K_1, K_2, K_3, Y)$ is defined as a quadruple consisting of three sets and a ternary relation $Y \subseteq K_1 \times K_2 \times K_3$. K_1 represents the set of objects, K_2 the set of attributes and K_3 the set of conditions. The notation for an element of the incidence relation is $(g, m, b) \in Y$ or $b(g, m)$ and it is read object g has attribute m under condition b.*

Polyadic FCA is a direct generalization of the dyadic or triadic case, where n (not necessarily different) non-empty sets are related via an n-ary relation. An n-concept is a maximal cluster of n sets, with every element being interrelated with all the others.

Definition 2. *Let $n \geq 2$ be a natural number. An* n-context *is an $(n+1)$-tuple $\mathbb{K} := (K_1, K_2, \dots, K_n, Y)$, where $K_1, K_2, \dots, K_n$ are sets and Y is an n-ary relation $Y \subseteq K_1 \times K_2 \times \dots \times K_n$.*

Definition 3. *The* n-concepts *of an n-context $(K_1, \dots, K_n, Y)$ are exactly the n-tuples $(A_1, \dots, A_n)$ that satisfy $A_1 \times \dots \times A_n \subseteq Y$ and which are maximal with respect to component-wise set inclusion. $(A_1, \dots, A_n)$ is called a* proper *n-concept if $A_1, \dots, A_n$ are all non-empty.*

Example 1. Finite dyadic contexts can be represented as cross-tables, rows being labeled with object names, columns with attribute names. In the triadic case, objects are related to attributes and conditions via a ternary relation and the corresponding triadic context can be thought of as a 3D cuboid, the ternary relation being marked by filled cells. Triadic contexts are usually unfolded into a series of dyadic "slices", like in the following example, where we consider a triadic context (K_1, K_2, K_3, Y) where the object set K_1 consists of authors of scientific papers, the attribute set K_2 contains conference names/journal names while the conditions K_3 are the publication years. For this small selection we obtain a $2 \times 4 \times 2$ triadic context, the "slices" being labeled by condition names (Fig. 1).

2014	*Corr*	*ICC*	*PIMRC*	*HICSS*
Rumpe	×			
Alouni	×	×	×	

2015	*Corr*	*ICC*	*PIMRC*	*HICSS*
Rumpe	×			×
Alouni	×	×		

Fig. 1. DBLP data: author, conference/journal, year

There are exactly six triconcepts of this context, i.e., maximal 3D cuboids full of incidences:

- $(\{Rumpe, Alouni\}, \{Corr\}, \{2014, 2015\})$,
- $(\{Alouni\}, \{Corr, ICC, PIMRC\}, \{2014\})$,
- $(\{Alouni\}, \{Corr, ICC\}, \{2014, 2015\})$,
- $(\{Rumpe\}, \{Corr, HICSS\}, \{2015\})$,
- $(\emptyset, \{Corr, ICC, PIMRC, HICSS\}, \{2014, 2015\})$ and
- $(\{Rumpe, Alouni\}, \{Corr, ICC, PIMRC, HICSS\}, \emptyset)$.

The first four of these triconcepts are called *proper*.

If $\mathbb{K} = (K_1, \dots, K_n, Y)$ is an n-context, *membership constraints* are indicating restricting conditions by specifying which specific elements $a_j \in K_j$ must be included in the jth component of an n-concept, respectively which elements $b_j \in K_j$, $j = 1, \ldots, n$ must be excluded therefrom. We investigated the question of satisfiability of such membership constraints, i.e., to determine if there are any formal n-concepts which are satisfying the inclusion and exclusion requirements [19].

Definition 4. *An* n-adic membership constraint *on an n-context* $\mathbb{K} = (K_1, \ldots, K_n, R)$ *is a $2n$-tuple* $\mathbb{C} = (K_1^+, K_1^-, \ldots, K_n^+, K_n^-)$ *with* $K_i^+ \subseteq K_i$ *called* required sets *and* $K_i^- \subseteq K_i$ *called* forbidden sets*.*

An n-concept $(A_1, \ldots, A_n)$ *of* $\mathbb{K}$ *is said to* satisfy *such a membership constraint if* $K_i^+ \subseteq A_i$ *and* $K_i^- \cap A_i = \emptyset$ *hold for all* $i \in \{1, \ldots, n\}$*.*

We let $\mathrm{Mod}(\mathbb{K}, \mathbb{C})$ *(*$\mathrm{Mod}_p(\mathbb{K}, \mathbb{C})$*) denote the set of all (proper) n-concepts of* $\mathbb{K}$ *that satisfy* $\mathbb{C}$*.*

An n-adic membership constraint $\mathbb{C}$ *is said to be* (properly) satisfiable *with respect to* $\mathbb{K}$*, if it is satisfied by one of its (proper) n-concepts, that is, if* $\mathrm{Mod}(\mathbb{K}, \mathbb{C}) \neq \emptyset$ *(*$\mathrm{Mod}_p(\mathbb{K}, \mathbb{C}) \neq \emptyset$*).*

We have shown that the problem of deciding satisfiability of a membership constraint w.r.t. an n-context is NP-complete in general [19]. The intractability easily carries over to proper satisfiability.

2.2 Answer Set Programming

Answer Set Programming (ASP) [7] is a logic programming formalism and hence uses a declarative approach to solve NP-hard problems. As opposed to the imperative approach, where the programmer tells the computer what steps to follow in order to solve a problem, declarative approaches merely describe the problem while the process of solving is delegated to generic strategies applied by highly optimized ASP engines. Mainly, in ASP one has to express the problem in a logic programming format consisting of facts and rules, so that the solutions of the problem correspond to models of the logic program.

In what follows, we briefly introduce the syntax and semantics of normal logic programs under the stable model semantics [7,10]. We will present directly the syntax used in the source code in order to avoid a translation phase from one syntax to the other.

Let $\mathcal{D}$ denote the *domain*, i.e. a countable set of elements, also called *constants*. Next, we define an *atom* as an expression of the type $\mathtt{p}(t_1, \ldots, t_n)$, where $\mathtt{p}$ is a *predicate* of arity $n \geq 0$ and every t_i is an element from the domain or a variable, denoted by an upper case letter. An atom is called *ground* if it is variable-free. The set of all ground atoms over $\mathcal{D}$ is denoted by $\mathcal{G}_{\mathcal{D}}$. A *(normal) rule* ρ is defined as:

$$a_1 \leftarrow b_1 \wedge \ldots \wedge b_k \wedge {\sim} b_{k+1} \wedge \ldots \wedge {\sim} b_m,$$

where $a_1, b_1, \ldots, b_m$ are atoms $m \geq k \geq 0$ with the observation that the left or the right part of the rule might be missing, but not both at the same time. The left part of the rule, i.e. the part before "$\leftarrow$" is called *head*, denoted $H(\rho) = \{a_1\}$, while the right part is the *body* of the rule, denoted $B(\rho) = \{b_1, \ldots b_k, \sim b_{k+1}, \ldots, \sim b_m\}$. As mentioned previously, a rule does not necessarily contain a non-empty head and body, namely, when the head of the rule is empty, we call the rule a *constraint* and, when the body of the rule is missing and a_1 is ground, it is called a *fact*. In case the rule is a fact, we usually omit the sign "$\leftarrow$" and write just the atom in the head of the rule. In the definition of the rule, "$\sim$" denotes default negation, which refers to the absence of information as opposed to classical negation "$\neg$" which implies that the negated information is present. Intuitively, "$\sim a$" means that $a \notin I$, while $\neg a$ implies $\neg a \in I$, for an interpretation I, where $I \subseteq \mathcal{G_D}$ can be understood as the set of ground atoms which are true. Furthermore, we denote $B^+(\rho) = \{b_1, \ldots, b_k\}$ and $B^-(\rho) = \{b_{k+1}, \ldots, b_m\}$. A rule ρ is called *safe* if each variable in ρ occurs in $B^+(\rho)$. Finally, we define a propositional normal logic program as a finite set of normal rules.

In order to define when a program Π is satisfied by an interpretation I, let $\mathcal{U}_\Pi$ denote the subset of constants from the domain that appear in the program Π and $Gr(\Pi)$ the grounded program, i.e. the set of grounded rules obtained by applying all the possible substitutions from the variables to the constants in $\mathcal{U}_\Pi$, for all the rules $\rho \in \Pi$. We say that an interpretation $I \subseteq \mathcal{G_D}$ satisfies a normal ground rule $\rho \in \Pi$ if and only if the following implication holds:

$$B^+(\rho) \subseteq I, B^-(\rho) \cap I = \emptyset \Rightarrow H(\rho) \subseteq I.$$

Then the interpretation I satisfies a non-ground rule if it satisfies all the possible groundings of the rule. Finally, the interpretation I satisfies a program Π if it satisfies all its rules, i.e. it satisfies the grounded program $Gr(\Pi)$.

An interpretation $I \in \mathcal{G_D}$ is called an *answer set* or a *stable model* [10] of the program Π if and only if it is the $\subseteq$-minimal model satisfying the reduct Π^I defined by $\Pi^I = \{H(\rho) : -B^+(\rho) \mid I \cap B^-(\rho), \rho \in Gr(\Pi)\}$. Furthermore, we define the set of *cautiously entailed* ground atoms as the intersection of all answer sets.

ASP solving is split in two phases. In the first phase, a grounder has to be used in order to process the logic program into a finite variable-free propositional representation of the problem encoding. In the next phase, a solver uses as input the output of the grounder and computes the solutions, i.e. the answer sets of the problem.

Researchers from the University of Potsdam developed a tool suite called Potassco[1], which is a collection of answer set solving software. In our research we used the solving tools from the this collection [8], since it is currently the most prominent solver leading in the latest competitions [1]. In what follows, we will describe these tools in more details.

The first tool, Gringo is a grounder that can be used in the first phase in order to transform the initial encoding into an equivalent variable-free, i.e. ground, program. Gringo has a simple, but comprehensive syntax that can express different types of rules (normal, choice, cardinality, etc.), constraints (integrity, cardinality, etc.), but also optimization statements. Intuitively, a constraint expresses a

[1] http://potassco.sourceforge.net/.

"forbidden" behavior of the models, i.e. if the body of the constraint is true then the model is not a stable model. The output of Gringo is in the *smodels* format, which is an intermediate format used for the ASP solver input.

The second tool, Clasp, is a solver that uses the smodels format for the input and computes the answer sets of the program. The output of Clasp can be configured by the user and shows all or some of the following details: the number of solutions and whether the problem is *satisfiable* (i.e. if it has at least one stable model) or *unsatisfiable*, the solutions, and the detailed time of the execution. The format of the answer sets is also configurable by specifying in the encoding which predicates shall be printed in the output.

Both tools, Gringo and Clasp, were combined into one tool, called Clingo, in order to avoid processing the ASP program with Gringo and then further process the output with Clasp. Avoiding the intermediate step is particularly useful if the grounded program is not of interest and the only results needed are the answer sets of the program. Furthermore, in case one needs to compute the execution time of the whole ASP solving process, the integrated tool shows the cumulative duration of the two phases, grounding and solving.

3 Encoding Membership Constraints in Answer Set Programming

Given that satisfiability of membership constraints can in general be NP-complete, it is nontrivial to find efficient algorithms for computing membership-constraint-satisfying concepts. We note here that the problem can be nicely expressed in answer set programming using an encoding introduced in a previous paper [19]. We will consider the n-adic case. Given an n-adic membership constraint $\mathbb{C} = (K_1^+, K_1^-, \ldots, K_n^+, K_n^-)$ on an n-context $\mathbb{K} = (K_1, \ldots, K_n, R)$, we represent the specific problem by the following set of ground facts $F_{\mathbb{K},\mathbb{C}}$:

- $\mathtt{set}_i(a)$ for all $a \in K_i$,
- $\mathtt{rel}(a_1, \ldots, a_n)$ for all $(a_1, \ldots, a_n) \in R$,
- $\mathtt{required}_i(a)$ for all $a \in K_i^+$, and
- $\mathtt{forbidden}_i(a)$ for all $a \in K_i^-$.

Now we let P denote the following fixed answer set program (with rules for every $i \in \{1, \ldots, n\}$):

$$\begin{aligned}
\mathtt{in}_i(x) &\leftarrow \mathtt{set}_i(x) \wedge \sim\mathtt{out}_i(x) \\
\mathtt{out}_i(x) &\leftarrow \mathtt{set}_i(x) \wedge \sim\mathtt{in}_i(x) \\
&\leftarrow \textstyle\bigwedge_{j\in\{1,\ldots,n\}} \mathtt{in}_j(x_j) \wedge \sim\mathtt{rel}(x_1, \ldots, x_n) \\
\mathtt{exc}_i(x_i) &\leftarrow \textstyle\bigwedge_{j\in\{1,\ldots,n\}\setminus\{i\}} \mathtt{in}_j(x_j) \wedge \sim\mathtt{rel}(x_1, \ldots, x_n) \\
&\leftarrow \mathtt{out}_i(x) \wedge \sim\mathtt{exc}_i(x) \\
&\leftarrow \mathtt{out}_i(x) \wedge \mathtt{required}_i(x) \\
&\leftarrow \mathtt{in}_i(x) \wedge \mathtt{forbidden}_i(x)
\end{aligned}$$

Intuitively, the first two lines “guess” an n-concept candidate by stipulating for each element of each K_i if they are in or out. The third rule eliminates a candidate if it violates the condition $A_1 \times \ldots \times A_n \subseteq R$, while the fourth and fifth rule ensure the maximality condition for n-concepts. Finally, the sixth and the seventh rule eliminate n-concepts violating the given membership constraint.

There is a one-to-one correspondence between the answer sets X of $F_{\mathbb{K},\mathbb{C}} \cup P$ and the n-concepts of $\mathbb{K}$ satisfying $\mathbb{C}$ obtained as $(\{a \mid \mathtt{in}_1(a) \in X\}, \ldots, \{a \mid \mathtt{in}_n(a) \in X\})$. Consequently, optimized off-the-shelf ASP tools can be used for checking satisfiability but also for enumerating all satisfying n-concepts.

4 Navigation

In this section, we describe a strategy for navigating the space of proper n-concepts of an n-context.[2] The basic idea is to use intuitive representations of “subspaces” of the overall space by specifying which elements must be included in or excluded from a certain proper n-concept component A_i. Obviously, such a subspace is identified by a membership constraint $\mathbb{C} = (K_1^+, K_1^-, \ldots, K_n^+, K_n^-)$ specifying exactly the included and excluded elements for each component of the n-concepts. The n-concepts in the “subspace” associated with $\mathbb{C}$ are then the n-concepts from $\mathrm{Mod}_p(\mathbb{K}, \mathbb{C})$. Visually, $\mathbb{C}$ can be represented by displaying $K_1, \ldots, K_n$ as n lists and indicating for every element if it is included, excluded, or none of the two (undetermined). The user can then choose to restrict the space further by indicating for an undetermined element of some K_i, if it should be included or excluded. What should, however be avoided is that by doing so, the user arrives at an empty “subspace”, i.e., a membership constraint that is not satisfied by any proper n-concept (i.e., $\mathrm{Mod}_p(\mathbb{K}, \mathbb{C}) = \emptyset$). To this end, we will update the membership constraint directly after the user interaction in order to reflect all necessary inclusions and exclusions automatically following from the user's choice. Assume $\mathbb{C} = (K_1^+, K_1^-, \ldots, K_n^+, K_n^-)$ is the membership constraint after the user interaction. The updated constraint can be described by $\mathbb{C}' = (L_1^+, L_1^-, \ldots, L_n^+, L_n^-)$, where

$$L_i^+ = \bigcap_{(A_1,\ldots,A_n)\in\mathrm{Mod}_p(\mathbb{K},\mathbb{C})} A_i$$

and

$$L_i^- = \bigcap_{(A_1,\ldots,A_n)\in\mathrm{Mod}_p(\mathbb{K},\mathbb{C})} K_i \setminus A_i.$$

It is then clear that after such an update, for every element e of some K_i which is still undetermined by $\mathbb{C}'$, there exist proper n-concepts $(E_1, \ldots, E_n)$ and $(F_1, \ldots, F_n)$ in $\mathrm{Mod}_p(\mathbb{K}, \mathbb{C}')$ with $e \in E_i$ but $e \notin F_i$. Consequently, whatever undetermined element the user chooses to include or exclude, the resulting

[2] Non-proper concepts are considered out of scope for knowledge exploration, thus we exclude them from our consideration. The described navigation would, however, also work if these concepts were taken into account.

membership constraint will be properly satisfiable. If the updated constraint $\mathbb{C}' = (L_1^+, L_1^-, \ldots, L_n^+, L_n^-)$ determines for every element if it is included or excluded (i.e., if $L_i^+ \cup L_i^- = K_i$ holds for every i), the user's navigation has narrowed down the space to the one proper n-concept $(L_1^+, \ldots, L_n^+)$.

Considering the example from the previous section, assume the user has specified the inclusion of the attribute *Corr* in K_2 and the exclusion of the attribute *ICC* from K_2, i.e.,

$$\mathbb{C} = (\emptyset, \emptyset, \{Corr\}, \{ICC\}, \emptyset, \emptyset).$$

The proper 3-concepts of $\mathbb{K}$ satisfying $\mathbb{C}$ are

$$\begin{aligned} C_1 &= (\{Rumpe, Alouni\}, \{Corr\}, \{2014, 2015\}) \quad \text{and} \\ C_2 &= (\{Rumpe\}, \{Corr, HICSS\}, \{2015\}), \end{aligned}$$

therefore, we would obtain the updated constraint

$$\mathbb{C}' = (\{Rumpe\}, \emptyset, \{Corr\}, \{ICC, PIMRC\}, \{2015\}, \emptyset).$$

If the user now decided to additionally exclude 2014 from K_3, leading to the constraint

$$\mathbb{C}'' = \quad (\{Rumpe\}, \emptyset, \{Corr\}, \{ICC, PIMRC\}, \{2015\}, \{2014\}),$$

the only proper 3-concept satisfying it is C_2. Consequently, $\mathbb{C}''$ will be updated to

$$\mathbb{C}''' = (\{Rumpe\}, \{Alouni\}, \{Corr, HICSS\}, \{ICC, PIMRC\}, \{2015\}, \{2014\}),$$

which then represents the final state of the navigation.

5 Implementation

Following the general scheme described in the previous section, we implemented a navigation tool for the cases $n \in \{2, 3, 4\}$, using different strategies for $n = 3$. The two fundamentally different approaches differ in the method of computing the concepts (ASP vs different tool), as well as in which navigation step the concepts are computed.

In 2015, we proposed the ASP encoding for the membership constraint satisfiability problem presented in Sect. 3 and discussed how it could be deployed in an interactive search scenario [19]. For the first approach presented in this paper[3], we extended and implemented this scenario using diverse ASP optimization techniques. For grounding and solving in the ASP navigation tool, we used Clingo from the Potassco collection [8] for the reasons mentioned in Sect. 2.2.

Recall that ASP solves a search problem by computing answer sets, which represent the models of a given answer set program (the so-called stable models)

[3] https://sourceforge.net/projects/asp-concept-navigation.

[7,9,11,17,18]. Our encoding, as presented in Sect. 3, is such that given $\mathbb{K}$ and $\mathbb{C}$, an answer set program is created, such that there is a one-to-one correspondence between the answer sets and the n-concepts of $\mathbb{K}$ satisfying $\mathbb{C}$.

The known facts in a membership constraint satisfiability problem are the elements of the context $K_i, i \in \{1, \ldots, n\}$, the n-adic relation Y and the sets of required and forbidden elements. The answer set program can be conceived as a declarative implementation of the following "guess & check" strategy:

- start from an empty constraint $\mathbb{C}$
- decide for each element $a \in K_i, i \in \{1, \ldots, n\}$, if $a \in K_i^+$, i.e. included, or $a \in K_i^-$, i.e. excluded, hence reaching a membership constraint of the form $\mathbb{C} = (K_1^+, K_1^-, \ldots, K_n^+, K_n^-)$ with $K_i^+ \cup K_i^- = K_i$ for every i
- check if $(K_1^+, \ldots, K_n^+)$ is component-wise maximal w.r.t. $K_1^+ \times K_2^+ \times \ldots \times K_n^+ \subseteq Y$
- check if the required and forbidden elements are assigned correspondingly in the obtained membership constraint C, i.e. required elements belong to K_i^+, forbidden elements belong to K_j^-, for some $i, j \in \{1, \ldots, n\}$.

At any step, if one of the conditions is violated, the membership constraint is eliminated from the set of models. Hence, in the end we obtain all the membership constraints that correspond to n-concepts satisfying the given restrictions. The ASP encoding can be easily extended to retrieve only the proper n-concepts satisfying $\mathbb{C}$, by adding an additional check that ensures $|K_i^+| > 0$ for every i.

Recall that the *cautious* option of ASP iteratively computes the intersection over all answer sets, in the order in which they are computed by the ASP solver. However, regardless of the ASP solver, the last outputted solution when using the *cautious* option is always the intersection of all answer sets of the program. Later in this section, we show how this option can be utilized to optimize the propagation phase of the navigation and, in the evaluation section, we present statistics showing that it has a great impact on the execution time of the tool we developed.

The propagation algorithm tests for all elements that are still in an undetermined state, which of the possible decisions on that element (*in* or *out*) give rise to a satisfiable answer set program. In case one of the decisions generates an unsatisfiable problem, the complementary choice is automatically made. Remember that, as discussed in the previous section, when starting from a satisfiable setting, it cannot be the case that both choices generate an unsatisfiable program.

The alternative to explicitly testing all the possible choices for every element in an undetermined state is to compute all the answer sets for the already added constraints and to obtain their intersection. This intersection contains the *in* and *out* choices that need to be propagated, since their complementary constraints are not included in any answer set and hence, would generate an unsatisfiable program. This approach is formally described in Algorithm 1.

Algorithm 1. propagation of user decisions optimized

```
function PROPAGATEOPTIMIZED($\mathbb{K}, \mathbb{C}$)
 Input: $n$-context $\mathbb{K}$, membership constraint $\mathbb{C}$
 Output: updated membership constraint
 Data: membership constraint $\mathbb{C} = (K_1^+, K_1^-, \ldots, K_n^+, K_n^-)$

    for all $i \in \{1, \ldots, n\}$ do
        $L_i^+ = \bigcap_{(A_1,\ldots,A_n) \in \mathrm{Mod}_p(\mathbb{K},\mathbb{C})} A_i$
        $L_i^- = \bigcap_{(A_1,\ldots,A_n) \in \mathrm{Mod}_p(\mathbb{K},\mathbb{C})} K_i \setminus A_i.$
    end for
    $\mathbb{C} = (L_1^+, L_1^-, \ldots, L_n^+, L_n^-)$
    return $\mathbb{C}$
end function
```

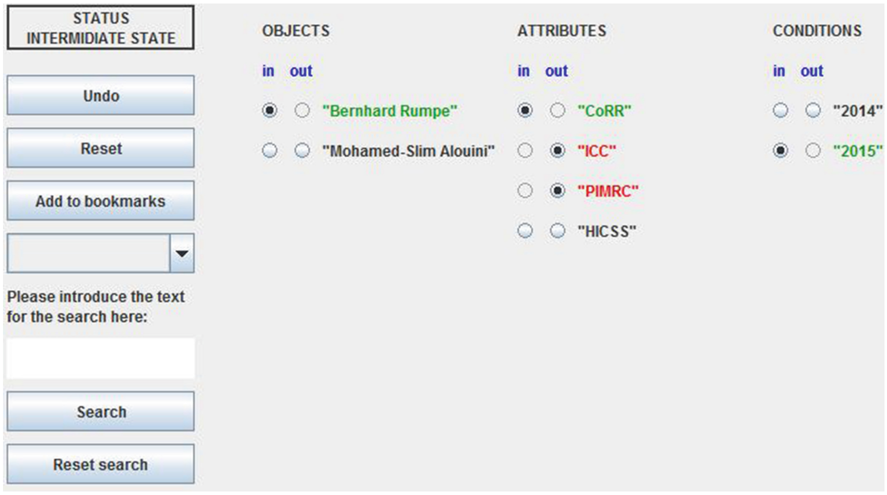
Algorithm 1 was implemented in the ASP navigation tool using the *cautious* option described previously. The implementation requires a single call to the ASP solver which computes the intersection of the models in the answer set. This intersection actually corresponds to the membership constraint containing all the inclusions and exclusions that need to be propagated. In comparison, for the simple propagation algorithm, multiple calls to the ASP solver are necessary: For each element that is in an undetermined state, two membership constraint satisfiability problems are generated, checking whether adding the element to the required objects, respectively to the forbidden objects, generates an unsatisfiable program. The cautious-based optimized propagation algorithm proved to drastically decrease the computation time as well as the memory usage, hence improving the performance of the interactive navigation tool. The experimental results are described in more detail in the evaluation section.

The optimized ASP approach was implemented and evaluated for triadic data. Furthermore, to show that it is easily extended to any n-adic context, we also implemented the dyadic and tetradic case, however, without evaluating their performance on real data sets. In fact, the only modifications that need to be made when updating the context's dimension are to add the new sets to the ASP encoding and to update the graphical interface and the context loader to the new context dimension.

The second approach for the navigation is a brute force implementation[4] and uses an exhaustive search in the whole space of n-concepts. Hence, a prerequisite for this tool is to previously compute all n-concepts using an existing tool. We implemented the triadic case ($n = 3$) and used Trias [13] to compute the triadic formal concepts. For that reason, the input for the navigation tool is adapted to Trias' output format.

This approach follows the same steps described in Algorithm 1, however it uses different methods for implementing them. The first main difference lies in

[4] https://sourceforge.net/projects/brute-force-concept-navigation.



STATUS
INTERMIDIATE STATE

OBJECTS

"Bernhard Rumpe": [✓ in] out
"Mohamed-Slim Alouini": in out

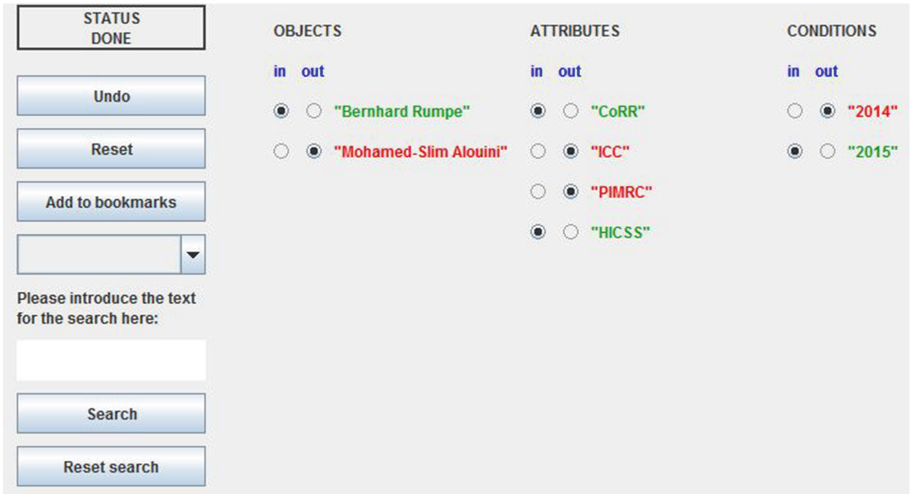
ATTRIBUTES

"CoRR": [✓ in] out
"ICC": in [✓ out]
"PIMRC": in [✓ out]
"HICSS": in out

CONDITIONS

"2014": in out
"2015": [✓ in] out

Fig. 2. Screenshot navigation tool: intermediate state (Color figure online)



STATUS
DONE

OBJECTS

"Bernhard Rumpe": [✓ in] out
"Mohamed-Slim Alouini": in [✓ out]

ATTRIBUTES

"CoRR": [✓ in] out
"ICC": in [✓ out]
"PIMRC": in [✓ out]
"HICSS": [✓ in] out

CONDITIONS

"2014": in [✓ out]
"2015": [✓ in] out

Fig. 3. Screenshot navigation tool: final state

the method of computing the triconcepts. Instead of computing them at each step using a declarative approach, in the brute force approach, all triconcepts are computed in the preprocessing phase using an existing algorithm. In a navigation step an exhaustive search is necessary in order to select the subset of triconcepts that satisfy the constraints and compute the intersection. This subset of triconcepts is successively pruned in each navigation step until it contains a single triconcept, which represents the final state of the navigation.

The graphical interface is the same for all implementations. The first column includes possible actions and information about the state of the navigation process (*intermediate* or *final*). The next columns each correspond to one dimension of the context and contain a list of the elements, each having two options next to it *in* or *out*. Figure 2 depicts a screenshot of the navigation example described in Sect. 4. It corresponds to the post propagation constraint $\mathbb{C}' = (\{Rumpe\}, \emptyset, \{Corr\}, \{ICC, PIMRC\}, \{2015\}, \emptyset)$. This is an intermediate state, where required elements are marked with green, forbidden elements with red, while elements in an undetermined state are unmarked. Furthermore, required and forbidden elements have the *in*, respectively the *out* column checked. Figure 3 shows the final state of the navigation, that corresponds to the membership constraint $\mathbb{C}''' = (\{Rumpe\}, \{Alouni\}, \{Corr, HICSS\}, \{ICC, PIMRC\}, \{2015\}, \{2014\})$.

6 Evaluation

In order to evaluate the implemented tools, we ran experiments on the dblp database[5]. The dblp database indexes conference and journal publications and contains information such as author, title, year, volume, and journal/conference name. In order to compare the ASP-based approach to the implemented brute force navigation, triadic datasets are needed. The triadic structure that we chose for the experiments contains the author's name, conference/journal name and year of the publication. We extracted the described triadic dataset from the dblp mysql dump and selected subsets of different dimensions. The subsets were selected by imposing restrictions on the number of publications per journal/conference, publication year and number of publications per author. For example, the dataset with 28 triples was obtained by the following steps:

– eliminate all journals/conferences having less than 15000 publications
– eliminate all publications before the year 2014
– eliminate all entries for authors that published less than 150 papers

After selecting a triadic data subset, no preprocessing phase for the ASP navigation tool is needed, since its input must contain only the triadic relation. However, the brute force navigation tool requires a preprocessing phase. First the triconcept set needs to be computed with the Trias algorithm, hence the Trias tool[6] needs to be installed separately. If using the Trias algorithm without a database connection, the standard input file requires numeric data. Hence, in order to format the data according to the Trias tool input format, the elements of the dataset need to be indexed. After running Trias to obtain the triconcepts, the output needs to be formatted again before using the brute force navigation tool. Mainly the dimensions and encodings of the object, attribute and condition sets need to be added, so that the navigation tool can output the names of

5 http://dblp.uni-trier.de/.
6 https://github.com/rjoberon/trias-algorithm.

Table 1. ASP navigator experiments

Number of objects	Number of attributes	Number of conditions	Number of triples	ASP navigation data loading time (s)	ASP navigation average step time (s)
2	15	2	28	0.015	0.1873
14	62	5	680	0.109	0.2315
41	67	7	2514	0.374	0.3278
68	67	8	4478	0.546	0.5930
83	67	9	5987	0.660	0.6350
108	67	10	8133	1.070	1.1940

Table 2. Brute force navigator experiments

Number of objects	Number of attributes	Number of conditions	Number of triples	Trias pre-processing time (s)	Brute force data loading time (s)	Brute force average step time (s)
2	15	2	28	0.27	0.016	0.0060
14	62	5	680	1.04	0.421	0.0047
41	67	7	2514	23.24	1.950	0.0219
68	67	8	4478	644.758	4.384	0.0530
83	67	9	5987	2152.839	6.992	0.1600
108	67	10	8133	>2h		

the elements and not their indexes. Only after these preprocessing steps can a user interactively navigate in the tricontext using the brute force navigation tool. Obviously, different formats for the input of the navigation tool can be implemented, but for the purpose of comparing the two tools we implemented one single input format based on the standard Trias output.

For measuring the runtimes of the two navigation tools, we have evaluated their performance on six different datasets containing between 28 and 8133 triples. The datasets are described in Tables 1 and 2 (we ran both tools on the same datasets), where objects are identified with author names, attributes with conferences/journal names and conditions with the publication years. For each dataset, we chose some random navigation paths through the data, which contain between 4 and 13 navigation steps and end when a final state, i.e., a triconcept, is reached. By navigation step we understand not only the action of a user choosing an element as *in* or *out*, but also the subsequent propagation phase. In order to compare the two approaches we computed the average navigation step time for each dataset and measured the time used for loading the data. This information can be obtained from the file *statistics.log* which is created as an output by the navigation tools. Furthermore, for the brute force navigation we also measured the preprocessing time, i.e. the time that Trias needs to compute the triconcepts. Note that the time needed to index the dataset for the Trias

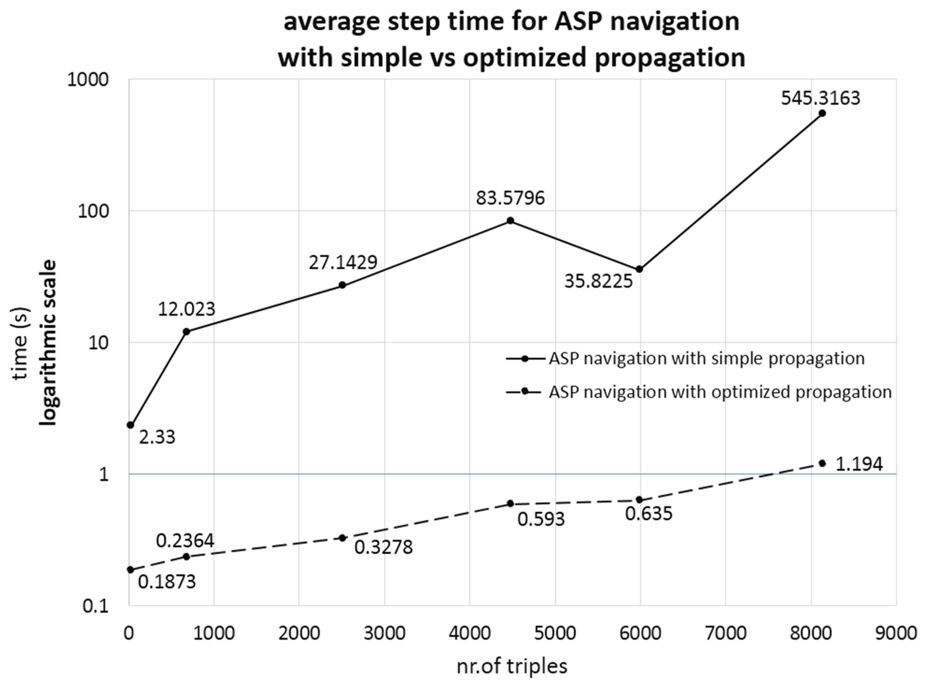


Fig. 4. Average step time for ASP navigation with simple propagation vs optimized propagation

input, as well as to add the encodings to the Trias output to obtain the input for the navigation tool, were excluded from this analysis, since this processing phase can be avoided by implementing different input/output formats for the Trias tool or for the brute force navigation tool. We denote the data loading time plus the preprocessing time as offline time. In case of the ASP navigation tool, the offline time equals the data loading time, since no preprocessing is needed. The experiments were run on an Intel(R) Core(TM) I7-3630QM CPU @ 2.40 GHz machine with 4 GB RAM and 6 M Cache.

First, we compared the different propagation implementations for the ASP approach: simple propagation vs. optimized propagation. The results are shown in Fig. 4, where the y-axis depicts the logarithmically scaled time of execution, while the x-axis corresponds to the size of the relation. Besides the big difference in the execution time of each step, the ASP navigation tool with simple propagation uses a lot of memory. For the context with 8133 triples after a few navigation steps the execution was stopped by the system because it reached the limit memory of 4 GB RAM. In comparison, this problem does not occur for the navigation tool with optimized propagation.

Next, we ran experiments on the same dataset to compare the ASP navigation tool with optimized propagation to the brute force navigation tool. Figure 5 shows the offline time of the ASP navigation tool vs. the brute force navigation

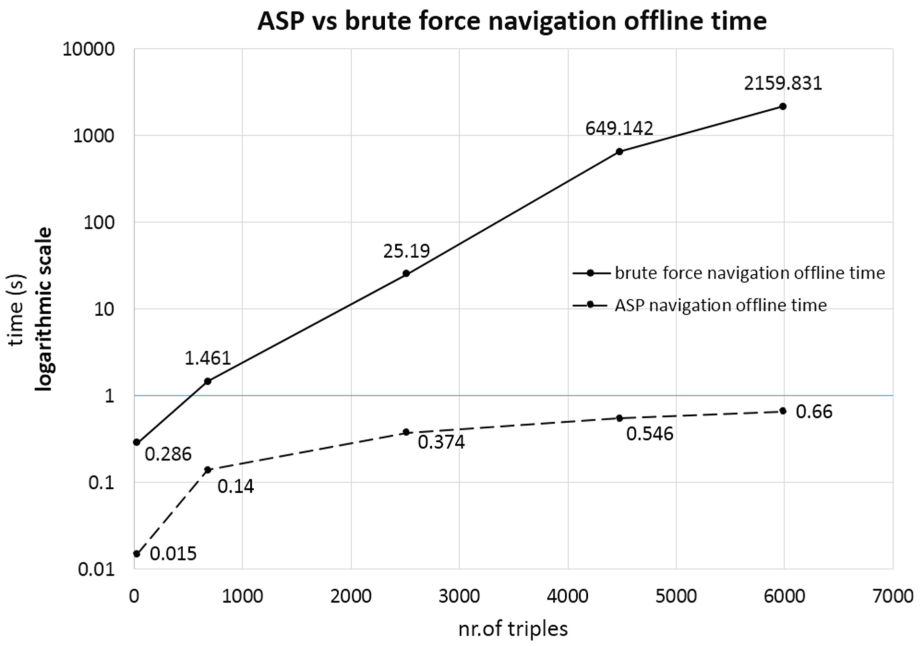


Fig. 5. Offline time for ASP vs brute force navigation tool with respect to the number of triples in the relation

tool on the logarithmically scaled y-axis in relation to the number of triples represented on the x-axis. As the chart shows, the offline time for the brute force navigation has a massive growth compared to the size of the triadic relation, while the offline time for the ASP navigation tool has a more linear growth. When comparing the average step time, the brute force navigation tool has slightly better results than the ASP navigation tool, but, as shown in Fig. 6, for subsets with less than 6000 triples the average step time is under 1 s for both approaches. Furthermore, from the experiments run on the larger data subset, containing 8133 triples, it followed that the ASP navigation tool is still usable, with an average step time of 1.194 s, as opposed to the brute force navigation tool, which turned out to have a very time consuming preprocessing phase: the Trias algorithm did not manage to compute the triconcept set in two hours.

The experiments lead us to believe that for larger datasets, the ASP navigation tool should be the preferred one, since it has a small execution time for loading the data, as well as for each navigation step, both of which are important for an interactive tool. Furthermore, in case of dynamic datasets that change frequently, it makes sense to use the ASP navigation tool which requires no preprocessing of the data.

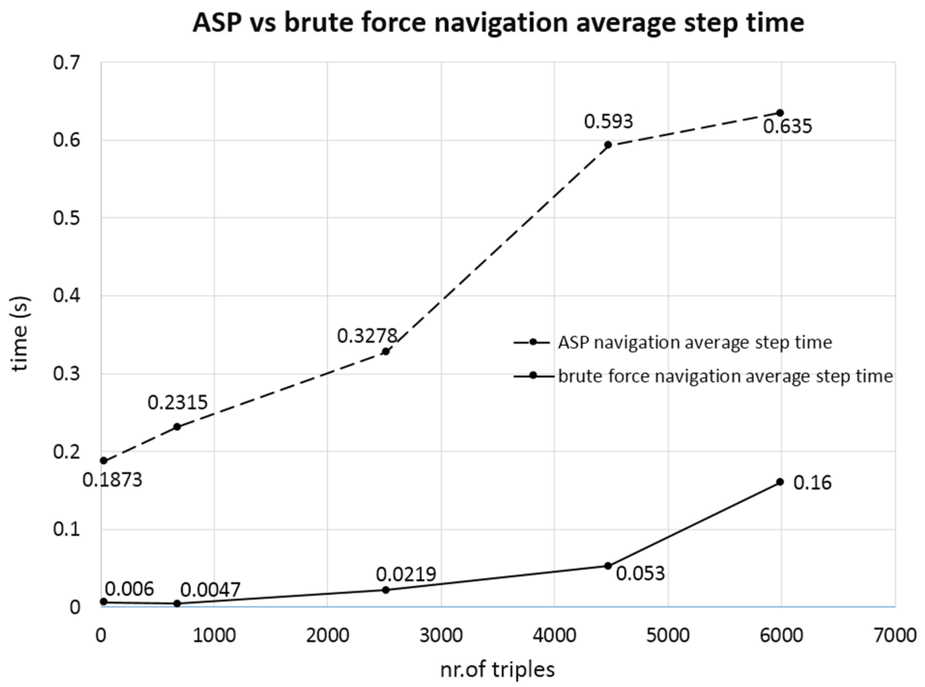


Fig. 6. Average step time for ASP vs brute force navigation tool with respect to the number of triples in the relation

7 Tool Extension

Having practical applications of FCA as a main motivation, we developed an extended tool, `FCA Tools Bundle` (for more details please refer to [15]), containing, besides basic FCA operations, the navigation method described in this paper as well as a different navigation method based on dyadic projections described in a previous paper [20]. From the different implementation approaches described in Sect. 5 we chose to include only the ASP based implementation in the new tool, since this method was evaluated as being more useful for large datasets.

The `FCA Tools Bundle`[7] currently implements features for dyadic and triadic formal concept analysis. The dyadic part contains features for lattice building and lattice visualization, while the triadic part focuses on navigation paradigms for tricontexts. The main purpose of the tool is to enable the user to visualize datasets of different dimensions as formal contexts and to interact with them. For this purpose, `FCA Tools Bundle` integrates different visualization formats offered by the formal contexts: concept list, concept lattice, local projections, particular formal concepts.

[7] https://fca-tools-bundle.com.

The graphical interface is similar to the one in the previous ASP navigation tool. For example, the same intermediate state represented in Fig. 2, can be seen in Fig. 7. In a similar manner, some of the elements are selected as being included or excluded, while the inclusions/exclusions of others (“Alouni”,“HICSS”,“2014”) remains to be determined in the next steps of the navigation. This fact is highlighted using the message box which assists the user during the navigation: “A concept was not yet pin-pointed but the constraints have been updated to prevent reaching an invalid state.”.

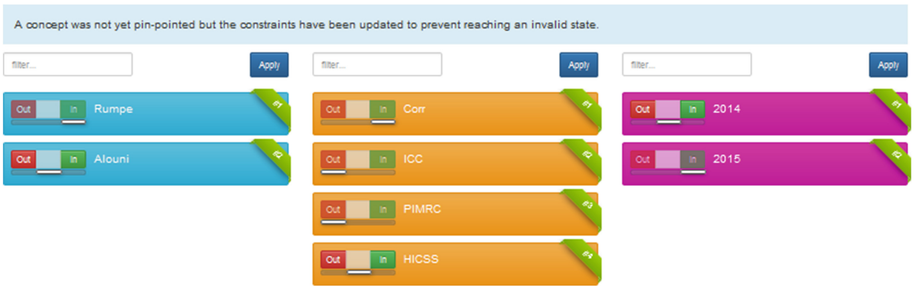
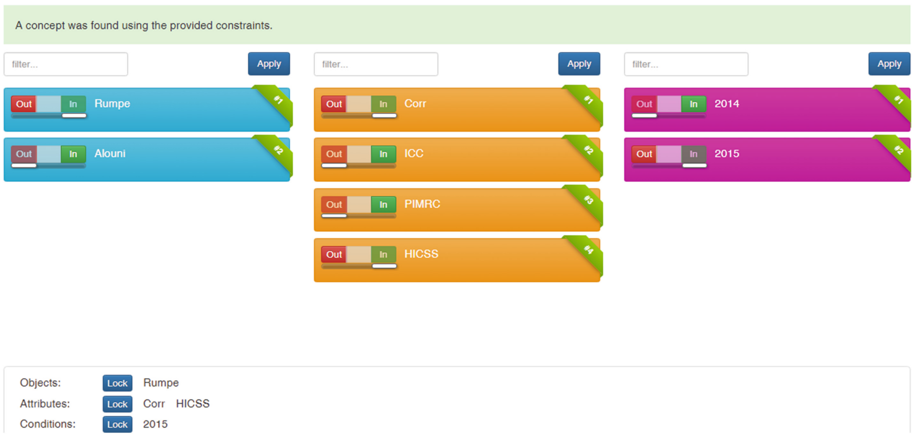

A concept was not yet pin-pointed but the constraints have been updated to prevent reaching an invalid state.

Fig. 7. Screenshot `FCA Tools Bundle`: intermediate state

In the final step of the navigation, the message shown to the user is updated accordingly: “A concept was found using the provided constraints.”. This can be observed in Fig. 8, which is the equivalent of Fig. 3 from the previous navigation tool. Moreover, the formal concept that was reached is represented separately on its three components: extent, intent and modus.


A concept was found using the provided constraints.

Objects: Lock Rumpe
Attributes: Lock Corr HICSS
Conditions: Lock 2015

Fig. 8. Screenshot `FCA Tools Bundle`: final state

`FCA Tools Bundle` usefully integrates different navigation methods proposed for triadic formal contexts. Therefore, one can use the ASP-based navigation in order to pin-point a single formal concept and then continue using the navigation based on dyadic projections for further exploration of the tricontext. This can be seen in Fig. 8, where before each component of the formal concept, there is a button "lock". This gives the user the possibility to project the context on that particular component and look at the corresponding dyadic concept lattice. Therefore, users can easily combine navigation methods to extensively explore a large dataset.

Another very important advantage of `FCA Tools Bundle` is the usability and accessibility of the tool. While one needs prior ASP and FCA knowledge in order to use the ASP navigation tool described in Sect. 5, there is no need for prior knowledge whatsoever in order to use `FCA Tools Bundle`. The user can simply import datasets in several formats and then explore them using the FCA- and ASP-enhanced methods.

8 Conclusion

This paper presents a navigation paradigm for polyadic FCA using different implementation strategies. For higher-adic FCA, this is, to the best of our knowledge, the first navigation tool which allows to explore, search, recognize, identify, analyze, and investigate polyadic concept sets by using membership constraints, in line with the Conceptual Knowledge Processing paradigm. Experiments on 3-dimensional datasets strongly suggest that the ASP navigation approach with optimized propagation is, in general, the better choice since it has low execution times, even for larger contexts. Furthermore, in case one needs to adapt the navigation tool to an n-dimensional context for $n \geq 5$, the ASP approach is easier generalized, by following the example of the already implemented cases $n \in \{2, 3, 4\}$, whereas for the brute force navigation approach, which was implemented only for $n = 3$ using Trias, one would first need to find an algorithm for computing the n-concepts. Moreover, we have presented a new tool, `FCA tools Bundle`, which allows users who are not familiar with FCA or ASP to analyze multi-dimensional datasets and browse through maximal clusters of data.

For future work, we intend to compare the ASP approach of the n-adic case with the naive approach that uses tools such as Data-Peeler [3] or Fenster [2] which claim to be able to compute closed patterns for n-adic datasets. Also, some new features or new implementations for existing features will be integrated in the `FCA tools Bundle`. Furthermore, are planning to further investigate exploration strategies and rule mining in polyadic datasets.

Acknowledements. Diana Troancă's research on this topic was supported by a scholarship from DAAD, the German Academic Exchange Service. The authors also thank the anonymous reviewers of an earlier version of this paper. We thank Lukas Schweizer for technical advice regarding ASP. The `FCA Tools Bundle` was implemented by Lorand Kis.

References

1. Calimeri, F., Gebser, M., Maratea, M., Ricca, F.: Design and results of the fifth answer set programming competition. Artif. Intell. **231**, 151–181 (2016)
2. Cerf, L., Besson, J., Nguyen, K., Boulicaut, J.: Closed and noise-tolerant patterns in n-ary relations. Data Min. Knowl. Discov. **26**(3), 574–619 (2013)
3. Cerf, L., Besson, J., Robardet, C., Boulicaut, J.F.: Closed patterns meet n-ary relations. ACM Trans. Knowl. Discov. Data **3**(1), 3:1–3:36 (2009)
4. Dragoş, S., Haliţă, D., Săcărea, C.: Behavioral pattern mining in web based educational systems. In: Rozic, N., Begusic, D., Saric, M., Solic, P. (eds.) Proceedings of the 23rd International Conference on Software, Telecommunications and Computer Networks (SoftCOM 2015), Split, Croatia, pp. 215–219. IEEE (2015)
5. Dragoş, S., Haliţă, D., Săcărea, C., Troancă, D.: Applying triadic FCA in studying web usage behaviors. In: Buchmann, R., Kifor, C.V., Yu, J. (eds.) KSEM 2014. LNCS (LNAI), vol. 8793, pp. 73–80. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-12096-6_7
6. Ganter, B., Wille, R.: Formal Concept Analysis - Mathematical Foundations. Springer, Heidelberg (1999). https://doi.org/10.1007/978-3-642-59830-2
7. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: Answer Set Solving in Practice. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan and Claypool Publishers, San Rafael (2012)
8. Gebser, M., Kaufmann, B., Kaminski, R., Ostrowski, M., Schaub, T., Schneider, M.T.: Potassco: The potsdam answer set solving collection. AI Commun. **24**(2), 107–124 (2011)
9. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: Kowalski, R.A., Bowen, K.A. (eds.) Logic Programming, Proceedings of the Fifth International Conference and Symposium, Seattle, Washington (2 Volumes), pp. 1070–1080. MIT Press (1988)
10. Gelfond, M., Lifschitz, V.: The Stable Model Semantics For Logic Programming. In: Kowalski, R., Bowen, K.A. (eds.) Proceedings of the Joint International Logic Programming Conference and Symposium, JICSLP 1988, Manchester, England, pp. 1070–1080. MIT Press (1988)
11. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Gener. Comput. **9**(3/4), 365–386 (1991)
12. Ignatov, D.I., Gnatyshak, D.V., Kuznetsov, S.O., Mirkin, B.G.: Triadic formal concept analysis and triclustering: searching for optimal patterns. Mach. Learn. **101**(1–3), 271–302 (2015)
13. Jäschke, R., Hotho, A., Schmitz, C., Ganter, B., Stumme, G.: TRIAS - an algorithm for mining iceberg tri-lattices. In: Clifton, C.W., Zhong, N., Liu, J., Wah, B.W., Wu, X. (eds.) Proceedings of the 6th IEEE International Conference on Data Mining (ICDM 2006), Hong Kong, China, pp. 907–911. IEEE Computer Society Press (2006)
14. Jäschke, R., Hotho, A., Schmitz, C., Ganter, B., Stumme, G.: Discovering shared conceptualizations in folksonomies. J. Web Semant. **6**(1), 38–53 (2008)
15. Kis, L.L., Sacarea, C., Troanca, D.: FCA tools bundle - A tool that enables dyadic and triadic conceptual navigation. In: Kuznetsov, S.O., Napoli, A., Rudolph, S. (eds.) Proceedings of the 5th International Workshop "What can FCA do for Artificial Intelligence?" co-located with the European Conference on Artificial Intelligence, FCA4AI@ECAI 2016, The Hague, The Netherlands. CEUR Workshop Proceedings, vol. 1703, pp. 42–50. CEUR-WS.org (2016)

16. Lehmann, F., Wille, R.: A triadic approach to formal concept analysis. In: Ellis, G., Levinson, R., Rich, W., Sowa, J.F. (eds.) ICCS-ConceptStruct 1995. LNCS, vol. 954, pp. 32–43. Springer, Heidelberg (1995). https://doi.org/10.1007/3-540-60161-9_27
17. Marek, V.W., Truszczyński, M.: Stable models and an alternative logic programming paradigm. In: Marek, V.M., Truszczyński, M., Warren, D.S. (eds.) The Logic Programming Paradigm: A 25-Year Perspective, pp. 375–398. Springer, Berlin, Heidelberg (1999). https://doi.org/10.1007/978-3-642-60085-2_17
18. Niemelä, I.: Logic programs with stable model semantics as a constraint programming paradigm. Ann. Math. Artif. Intell. **25**(3–4), 241–273 (1999)
19. Rudolph, S., Săcărea, C., Troancă, D.: Membership constraints in formal concept analysis. In: Yang, Q., Wooldridge, M. (eds.) Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI 2015), Buenos Aires, Argentina, pp. 3186–3192. AAAI Press (2015)
20. Rudolph, S., Săcărea, C., Troancă, D.: Towards a navigation paradigm for triadic concepts. In: Baixeries, J., Sacarea, C., Ojeda-Aciego, M. (eds.) ICFCA 2015. LNCS (LNAI), vol. 9113, pp. 252–267. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-19545-2_16
21. Săcărea, C.: Investigating oncological databases using conceptual landscapes. In: Hernandez, N., Jäschke, R., Croitoru, M. (eds.) ICCS 2014. LNCS (LNAI), vol. 8577, pp. 299–304. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08389-6_26
22. Voutsadakis, G.: Polyadic concept analysis. Order - A J. Theor. Ordered Sets Appl. **19**(3), 295–304 (2002)
23. Wille, R.: The basic theorem of triadic concept analysis. Order - A J. Theor. Ordered Sets Appl. **12**(2), 149–158 (1995)
24. Wille, R.: Methods of conceptual knowledge processing. In: Missaoui, R., Schmidt, J. (eds.) ICFCA 2006. LNCS (LNAI), vol. 3874, pp. 1–29. Springer, Heidelberg (2006). https://doi.org/10.1007/11671404_1