Conceptual Navigation for Polyadic Formal Concept Analysis*

Sebastian Rudolph[†] and **Christian Săcărea**[‡] and **Diana Troancă**[‡]

[†]Technische Universität Dresden, Germany
sebastian.rudolph@tu-dresden.de

[‡]Universitatea Babeş-Bolyai, 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 an extension of classical FCA that instead of a binary uses an n-ary incidence relation to define *formal concepts*, i.e. data clusters in which all elements are interrelated. We consider a paradigm for navigating the space of 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 optimization strategies offered by ASP. 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 argumentation and communication of why something happens in a particular way. Formal Concept Analysis (FCA) [Ganter and Wille, 1999] is a mathematical theory introduced by R. Wille, being the core of Conceptual Knowledge Processing [Wille, 2006]. 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 representation, acquiring, 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, F. Lehmann and R. Wille extended dyadic FCA to a triadic setting (3FCA) [Lehmann and Wille, 1995]; here *objects* are related to *attributes* under certain *conditions*. The *triadic concepts* 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 [Rudolph *et al.*, 2015b]. 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.

G. Voutsadakis [2002] 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 [1995].

FCA was successfully used on triadic or tetradic datasets such as folksonomies [Jäschke *et al.*, 2008], data logs of rental services [Cerf *et al.*, 2013] or data about mobile operators [Ignatov *et al.*, 2015]. 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 pur-

*Diana Troancă was supported by a scholarship from DAAD.

poses. Several strategies have been proposed to overcome this problem. For instance, Dragoş et al. [2014; 2015] are using a circular representation for triadic data while investigating users' behavioral patterns in e-learning environments. Săcărea [2014] 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 [Rudolph *et al.*, 2015a]. 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 ($n = 4$) 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

In this section, we briefly present the necessary basic notions and definitions. 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 1. *Let $n \geq 2$ be a natural number. An* n-context *is an $(n+1)$-tuple $\mathbb{K} := (K_1, K_2, \ldots, K_n, Y)$, where $K_1, K_2, \ldots, K_n$ are sets and Y is an n-ary relation $Y \subseteq K_1 \times K_2 \times \cdots \times K_n$.*

Definition 2. *The* n-concepts *of an n-context $(K_1, \ldots, K_n, Y)$ are exactly the n-tuples $(A_1, \ldots, A_n)$ that satisfy $A_1 \times \cdots \times A_n \subseteq Y$ and which are maximal with respect to component-wise set inclusion. $(A_1, \ldots, A_n)$ is called a* proper *n-concept if $A_1, \ldots, A_n$ 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.*

2014	*Corr*	*ICC*	*PIMRC*	*HICSS*
Rumpe	×			
Alouni	×	×	×	
2015	***Corr***	***ICC***	***PIMRC***	***HICSS***
Rumpe	×			×
Alouni	×	×		

Figure 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 proper.

If $\mathbb{K} = (K_1, \ldots, K_n, Y)$ 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 [Rudolph *et al.*, 2015a].

Definition 3. *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 [Rudolph *et al.*, 2015a]. The intractability easily carries over to proper satisfiability.

3 Navigation

In this section, we describe a strategy for navigating the space of proper n-concepts of an n-context.[1] 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 $\text{Mod}_p(\mathbb{K}, \mathbb{C})$. 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., $\text{Mod}_p(\mathbb{K}, \mathbb{C}) = \emptyset$). 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

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

and

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

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 $\text{Mod}_p(\mathbb{K}, \mathbb{C}')$ with $e \in E_i$ but $e \notin F_i$. Consequently, whatever undetermined element the user chooses to include or exclude, the resulting 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

$C_1 = (\{Rumpe, Alouni\}, \{Corr\}, \{2014, 2015\})$ and
$C_2 = (\{Rumpe\}, \{Corr, HICSS\}, \{2015\})$,

therefore, we would obtain the updated constraint

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

[1] 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.

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

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

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.

4 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 an ASP encoding for the membership constraint satisfiability problem and described an interactive search scenario [Rudolph *et al.*, 2015a]. Currently, in our first approach[2], we extended and implemented this scenario using different ASP optimization techniques. For grounding and solving in the ASP navigation tool we used Clingo from the Potassco collection [Gebser *et al.*, 2011], since it is currently the most prominent solver leading the latest competitions [Calimeri *et al.*, 2016].

ASP solves a search problem by computing answer sets, which represent the models of a given answer set program (the so-called stable models) [Gebser *et al.*, 2012; Gelfond and Lifschitz, 1988; 1991; Marek and Truszczyński, 1999; Niemelä, 1999]. Our encoding 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 $i, j \in \{1, \ldots, n\}$

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

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 formal 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 counts if $|K_i^+|>0$ for every i.

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 used to optimize the propagation phase of the navigation.

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 in this section. 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 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, without however 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[3] and uses an exhaustive search in the whole formal concept space. Hence, a prerequisite for this tool is to previously compute all formal concepts using an existing tool. We implemented the triadic case and used Trias [Jäschke *et al.*, 2006] 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 the method of computing the formal concepts. Instead of computing them at each step using a declarative approach, in the brute force approach all formal concepts 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 formal concepts that satisfy the constraints and compute the intersection. This subset of formal concepts is successively pruned in each navigation step until it contains a single concept, which represents the final state of the navigation.

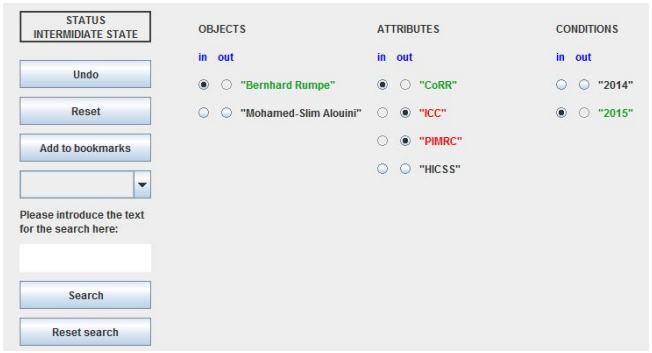

Figure 2: Screenshot navigation tool: intermediate state

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,

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

each having two options next to it *in* or *out*. Figure 2 depicts a screenshot of the navigation example described in section 3. 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.

5 Evaluation

In order to evaluate the implemented tools, we ran experiments on the dblp database[4]. 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 to the implemented brute force navigation tool one needs triadic datasets. 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 following the next 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[5] 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 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 Table 1, 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 formal concept, 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 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 6M Cache.

First we compared the different propagation implementations for the ASP approach: simple propagation vs. optimized propagation. The results are shown in Figure 3, 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 4 shows the offline time of the ASP navigation tool vs. the brute force navigation 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 Figure 5, for subsets with less than 6000 triples the average step time is under 1 second for both approaches. Furthermore, from the experiments ran 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 seconds, 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.

[4] http://dblp.uni-trier.de/

[5] https://github.com/rjoberon/trias-algorithm

Table 1: ASP and brute force Navigator experiments

object nr.	attribute nr.	condition nr.	triples nr.	ASP navigation data loading time (s)	ASP navigation average step time (s)	Trias preprocessing time (s)	brute force navigation data loading time (s)	brute force navigation average step time (s)
2	15	2	28	0.015	0.1873	0.27	0.016	0.006
14	62	5	680	0.109	0.2315	1.04	0.421	0.0047
41	67	7	2514	0.374	0.3278	23.24	1.95	0.0219
68	67	8	4478	0.546	0.593	644.758	4.384	0.053
83	67	9	5987	0.66	0.635	2152.839	6.992	0.16
108	67	10	8133	1.07	1.194	$>$ 2h		

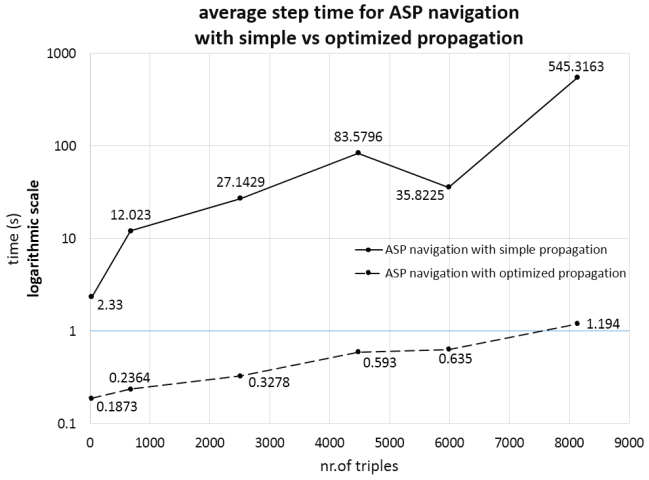

Figure 3: Average step time for ASP navigation with simple propagation vs optimized propagation

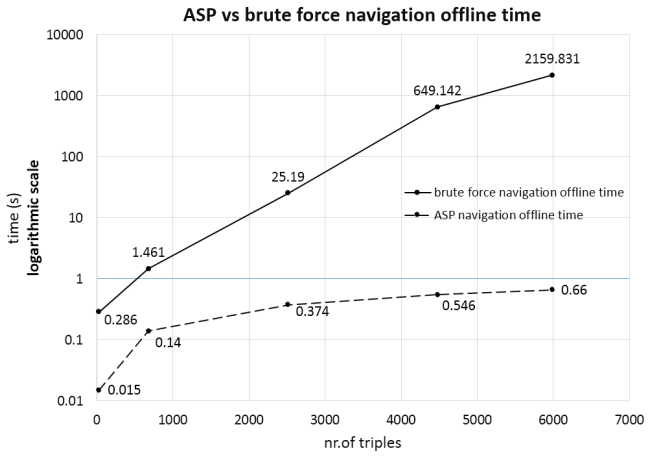

Figure 4: Offline time for ASP vs brute force navigation tool with respect to the number of triples in the relation

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.

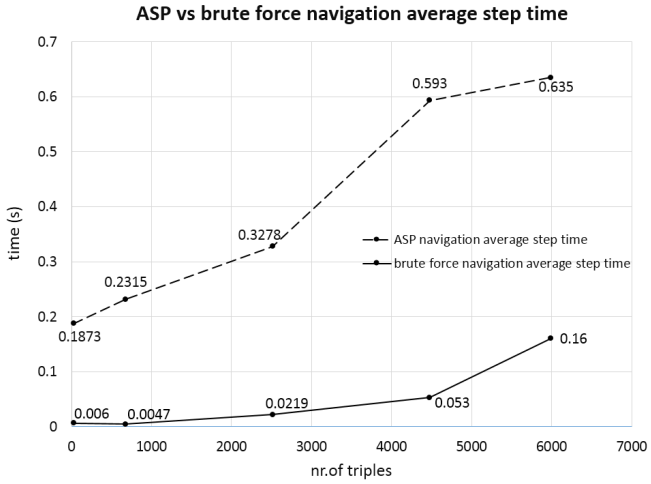

Figure 5: Average step time for ASP vs brute force navigation tool with respect to the number of triples in the relation

6 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 suggests 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.

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 [Cerf *et al.*, 2009] or Fenster [Cerf *et al.*, 2013] which claim to be able to compute closed patterns for n-adic datasets. Furthermore, we will focus on exploration strategies and rule mining in polyadic datasets.

References

[Calimeri *et al.*, 2016] Francesco Calimeri, Martin Gebser, Marco Maratea, and Francesco Ricca. Design and results of the fifth answer set programming competition. *Artif. Intell.*, 231:151–181, 2016.

[Cerf *et al.*, 2009] Loïc Cerf, Jérémy Besson, Céline Robardet, and Jean-François Boulicaut. Closed patterns meet n-ary relations. *ACM Trans. Knowl. Discov. Data*, 3(1):3:1–3:36, 2009.

[Cerf *et al.*, 2013] Loïc Cerf, Jérémy Besson, Kim-Ngan Nguyen, and Jean-François Boulicaut. Closed and noise-tolerant patterns in n-ary relations. *Data Min. Knowl. Discov.*, 26(3):574–619, 2013.

[Dragoş *et al.*, 2014] Sanda Dragoş, Diana Haliţă, Christian Săcărea, and Diana Troancă. Applying triadic FCA in studying web usage behaviors. In Robert Buchmann, Claudiu Vasile Kifor, and Jian Yu, editors, *Proceedings of the 7th Internetiontal Conference on Knowledge Science, Engineering and Management (KSEM 2014), Sibiu, Romania*, volume 8793 of *Lecture Notes in Computer Science*, pages 73–80. Springer, 2014.

[Dragoş *et al.*, 2015] Sanda Dragoş, Diana Haliţă, and Christian Săcărea. Behavioral pattern mining in web based educational systems. In Nikola Rozic, Dinko Begusic, Matko Saric, and Petar Solic, editors, *Proceedings of the 23rd International Conference on Software, Telecommunications and Computer Networks (SoftCOM 2015), Split, Croatia*, pages 215–219. IEEE, 2015.

[Ganter and Wille, 1999] Bernhard Ganter and Rudolf Wille. *Formal concept analysis - mathematical foundations*. Springer, 1999.

[Gebser *et al.*, 2011] Martin Gebser, Benjamin Kaufmann, Roland Kaminski, Max Ostrowski, Torsten Schaub, and Marius Thomas Schneider. Potassco: The potsdam answer set solving collection. *AI Commun.*, 24(2):107–124, 2011.

[Gebser *et al.*, 2012] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. *Answer Set Solving in Practice*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan and Claypool Publishers, 2012.

[Gelfond and Lifschitz, 1988] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In Robert A. Kowalski and Kenneth A. Bowen, editors, *Logic Programming, Proceedings of the Fifth International Conference and Symposium, Seattle, Washington (2 Volumes)*, pages 1070–1080. MIT Press, 1988.

[Gelfond and Lifschitz, 1991] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Comput.*, 9(3/4):365–386, 1991.

[Ignatov *et al.*, 2015] Dmitry I. Ignatov, Dmitry V. Gnatyshak, Sergei O. Kuznetsov, and Boris G. Mirkin. Triadic formal concept analysis and triclustering: searching for optimal patterns. *Machine Learning*, 101(1-3):271–302, 2015.

[Jäschke *et al.*, 2006] Robert Jäschke, Andreas Hotho, Christoph Schmitz, Bernhard Ganter, and Gerd Stumme. TRIAS - an algorithm for mining iceberg tri-lattices. In Christopher W. Clifton, Ning Zhong, Jiming Liu, Benjamin W. Wah, and Xidong Wu, editors, *Proceedings of the 6th IEEE International Conference on Data Mining (ICDM 2006), Hong Kong, China*, pages 907–911. IEEE Computer Society Press, 2006.

[Jäschke *et al.*, 2008] Robert Jäschke, Andreas Hotho, Christoph Schmitz, Bernhard Ganter, and Gerd Stumme. Discovering shared conceptualizations in folksonomies. *Journal of Web Semantics*, 6(1):38–53, 2008.

[Lehmann and Wille, 1995] Fritz Lehmann and Rudolf Wille. A triadic approach to formal concept analysis. In Gerard Ellis, Robert Levinson, William Rich, and John F. Sowa, editors, *Proceedings of the Third International Conference on Conceptual Structures (ICCS 1995), Santa Cruz, California, USA*, volume 954 of *Lecture Notes in Computer Science*, pages 32–43. Springer, 1995.

[Marek and Truszczyński, 1999] Victor W. Marek and Miroslaw Truszczyński. *The Logic Programming Paradigm: A 25-Year Perspective*, chapter Stable Models and an Alternative Logic Programming Paradigm, pages 375–398. Springer, Berlin, Heidelberg, 1999.

[Niemelä, 1999] Ilkka Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Ann. Math. Artif. Intell.*, 25(3-4):241–273, 1999.

[Rudolph *et al.*, 2015a] Sebastian Rudolph, Christian Săcărea, and Diana Troancă. Membership constraints in formal concept analysis. In Qiang Yang and Michael Wooldridge, editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI 2015), Buenos Aires, Argentina*, pages 3186–3192. AAAI Press, 2015.

[Rudolph *et al.*, 2015b] Sebastian Rudolph, Christian Săcărea, and Diana Troancă. Towards a navigation paradigm for triadic concepts. In Jaume Baixeries, Christian Săcărea, and Manuel Ojeda-Aciego, editors, *Proceedings of the 13th International Conference on Formal Concept Analysis (ICFCA 2015), Nerja, Spain*, volume 9113 of *Lecture Notes in Computer Science*, pages 252–267. Springer, 2015.

[Săcărea, 2014] Christian Săcărea. Investigating oncological databases using conceptual landscapes. In Nathalie Hernandez, Robert Jäschke, and Madalina Croitoru, editors, *Proceedings of the 21st Internation Conference on Conceptual Structures: Graph-Based Representation and Reasoning (ICCS 2014), Iaşi, Romania*, volume 8577 of *Lecture Notes in Computer Science*, pages 299–304. Springer, 2014.

[Voutsadakis, 2002] George Voutsadakis. Polyadic concept analysis. *Order - A Journal on The Theory of Ordered Sets and Its Applications*, 19(3):295–304, 2002.

[Wille, 1995] Rudolf Wille. The basic theorem of triadic concept analysis. *Order - A Journal on The Theory of Ordered Sets and Its Applications*, 12(2):149–158, 1995.

[Wille, 2006] Rudolf Wille. Methods of conceptual knowledge processing. In Rokia Missaoui and Jürg Schmid, editors, *Proceedings of the 4th International Conference on Formal Concept Analysis (ICFCA 2006), Dresden, Germany*, volume 3874 of *Lecture Notes in Computer Science*, pages 1–29. Springer, 2006.