A Tool for Creating and Visualising Formal Concept Trees

Simon Andrews and Laurence Hirsch

Conceptual Structures Research Group
Communication and Computing Research Centre
Faculty of Arts, Computing, Engineering and Sciences
Sheffield Hallam University, Sheffield, UK
`s.andrews@shu.ac.uk l.hirsch@shu.ac.uk`

Abstract. This paper presents a tool for creating and visualising formal concept trees. The concept tree provides an alternative visualisation to the more commonly known concept lattice. The tool described here is an extension of the In-Close formal concept mining program, where concepts are output in a format that can be visualised in a Web Browser using the Collapsible Tree Layout from the D3.js JavaScript library. Because the visualisation is expandable and collapsible, the tool is able to deal with large trees and the user is able to explore branches with single mouse clicks and by panning and zooming the tree. So-called 'iceberg trees' can also be produced, by specifying a minimum support for objects.

1 Introduction

There are a number of tools that visualise and explore formal concept lattices, such as the well known Concept Explorer [15] and others including ToscanaJ [4], Conflexplore [6], FcaStone/Graphviz [11] and Galicia [13]. However, there are, as far as we know, no similar tools to visualise and explore formal concept trees. Although the concept lattice is the natural and primary means of visualisation in FCA, the concept tree is a useful alternative, often easier to digest (particularly by non-FCA speakers) and particularly appropriate for situations where the underlying information structure is a tree, such as a taxonomy or a decision tree. Although there are several publications that show the creation and application of formal concept trees, such as [5,10,14], the visualisations therein have been created bespoke for the work concerned rather than generated by an automated tool.

There is also a problem in concept lattices regarding size and complexity: a lattice soon becomes too large and too complex to read and manage, or even compute, and although most of the lattice visualisation tools listed above have some means or other of attempting to address the problem (such as de-selecting attributes and objects in Concept Explorer), none of them are not really capable of handling a large number of concepts, either producing a 'bird's nest' of indecipherable nodes and arcs, or simply unable to compute the structure in the first place.

This paper presents a tool that creates formal concept trees from formal contexts and visualises them using the Collapsible Tree Format from the D3.js JavaScript library. The tree is created as an output from a modified version of the fast concept miner, In-Close2 [1,3,2]. The modification allows In-Close2 to output formal concepts in the D3 Collapsible Tree JSON file format, so that they can be visualised as a tree in a Web-Browser. With simple controls for expanding and collapsing nodes of the tree, the tool is easy to use and the trees produced are easy to explore. Also, because the tree is collapsible and expandable, and because In-Close2 is capable of quickly computing large numbers of concepts, it is possible to create, manage and explore large concept trees; the limitation being only on the JavaScript to deal with very large JSON files.

The following sections describe the use of these two components with some well known FCA examples as illustration.

2 Concept Trees

A tree, as a formal structure, is a set of nodes connected by arcs, such that each node has a single parent node. A lattice can be thought of as a collection of overlapping trees [5] where nodes with multiple parents have their connecting arcs reduced to leave single parents. In terms of connections between formal concepts in a concept lattice, where an arc is removed to form a corresponding tree, any inherited objects and attributes that are lost by removing the connection must be restored to the child and parent concepts whose connection has been removed. Thus the concepts in the lattice and the tree are exactly the same - only the number of connections have been reduced. Figure 1 is a simple example showing a concept lattice on the left and a corresponding concept tree on the right. The attributes in each case are $a1, a2, a3, a4$ and the objects $o1, o2, o3, o4$. Notice how attribute $a3$ and object $o4$ are labeled twice, to as to maintain the intents and extents of the concepts whose connection has been severed in the creation of the tree.

The latest version of In-Close, In-Close2.8, has the facility to output formal concept trees in JSON format for the D3.js Collapsible Tree Layout [1]. The input for the process is a formal context in the well-known Burmeister `cxt` format. The tree format is simply one of the output options presented to the user when In-Close2.8 is run. Figure 2 shows the Command Line interface with the well known formal context Live-in-water being used as an example.

It is worth noting that, although concept trees have been described above as originating from lattices that have had arcs removed, this is not how In-Close creates a concept tree. In-Close does not compute the lattice at all, it simply computes the first occurrence of each child concept, in the manner of Close-by-One type algorithms [9,8], and ignores subsequent computations of the same child, thus not creating any connections to further parents. Whereas previous publications concerning the creation of concept trees have discussed different

[1] `https://bl.ocks.org/mbostock/4339083`

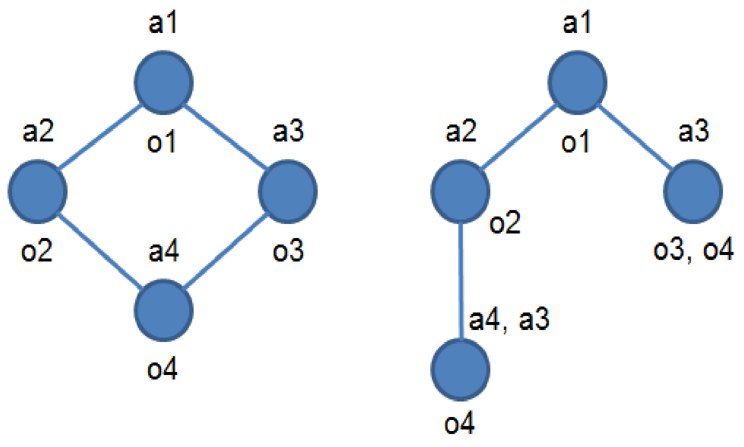


Fig. 1. A concept lattice (left) with a corresponding concept tree (right)

approaches to 'pruning' lattice arcs to make trees, using In-Close involves no such decision making - the tree created depends only on the order in which it computes formal concepts from the formal context it is provided with.

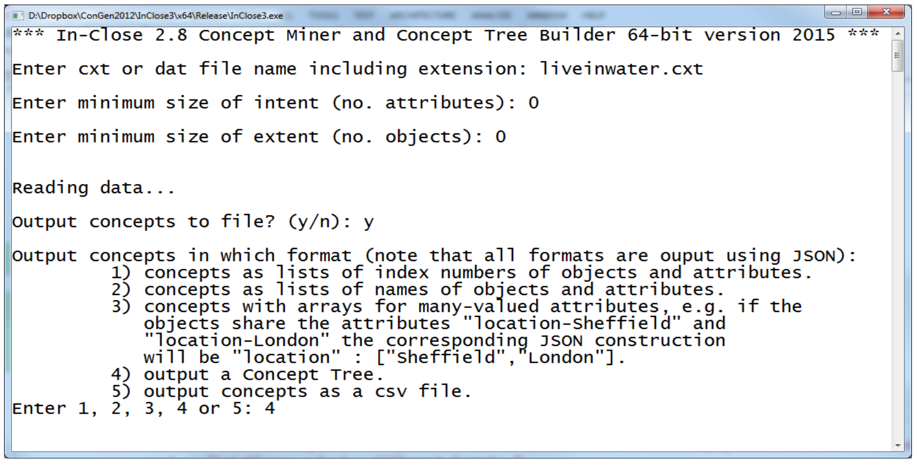
The concept tree is output as a file called `concepts.json` ready to be uploaded to the JavaScript program.

3 Visualising the Concept Tree

The JavaScript program can be opened in a browser that supports HTML5 (e.g. Chrome but not Internet Explorer). The `concepts.json` file is uploaded and visualised using the 'Choose File' and 'Upload->Draw Tree' buttons, see Figure 3. By default, the tree is initially displayed to the first level of concepts below the top concept. Figure 3 shows a concept tree created from the well-known Live-in-water context.

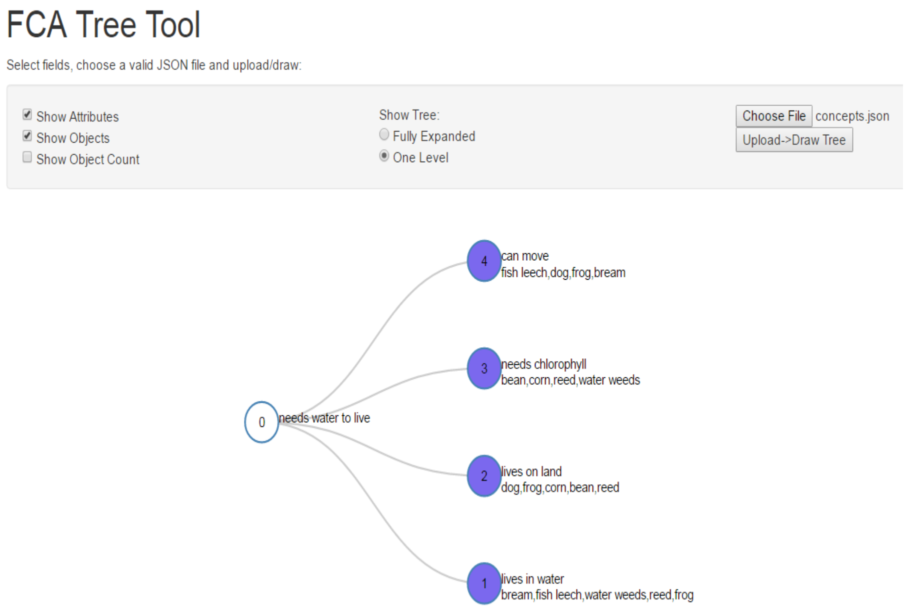
The concepts are numbered with the top concept being number 0. Concepts are labeled with attributes (upper line) and objects (lower line). If the concept is filled in with colour, it means that it can be expanded to another level in the tree by clicking the concept. Clicking concept 2 in the Live-in-water example expands it as shown in Figure 4, revealing more specialised sub-concepts that introduce more attributes. Notice that the objects 'move' from concept 2 to become labeled at the appropriate sub-concept. Thus 'frog' and 'dog' are now labeled at concept 8 since, as well as living on land, they can move and they have limbs.

Clicking on an un-filled concept will collapse the tree to that concept. Thus clicking now on concept 2 will collapse concepts 7 and 8 back up to it, restoring



```
*** In-Close 2.8 Concept Miner and Concept Tree Builder 64-bit version 2015 ***

Enter cxt or dat file name including extension: liveinwater.cxt

Enter minimum size of intent (no. attributes): 0

Enter minimum size of extent (no. objects): 0


Reading data...

Output concepts to file? (y/n): y

Output concepts in which format (note that all formats are ouput using JSON):
         1) concepts as lists of index numbers of objects and attributes.
         2) concepts as lists of names of objects and attributes.
         3) concepts with arrays for many-valued attributes, e.g. if the
            objects share the attributes "location-Sheffield" and
            "location-London" the corresponding JSON construction
            will be "location" : ["Sheffield","London"].
         4) output a Concept Tree.
         5) output concepts as a csv file.
Enter 1, 2, 3, 4 or 5: 4
```

Fig. 2. Creating a Concept Tree using In-Close2.8


FCA Tree Tool

Select fields, choose a valid JSON file and upload/draw:

[✓] Show Attributes
[✓] Show Objects
[ ] Show Object Count

Show Tree:
( ) Fully Expanded
(●) One Level

Choose File concepts.json
Upload->Draw Tree

Fig. 3. Visualising a Concept Tree in a Web-Browser

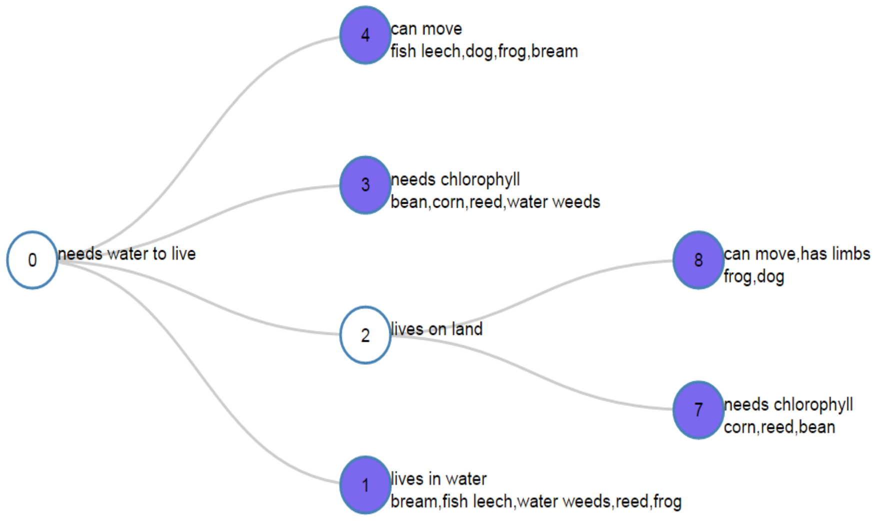


Fig. 4. Expanding Live-in-water Concept 2

concept 2 to its state in Figure 3. Clicking on the top concept (concept 0) will collapse the tree to a single node.

To aid the user in exploring and managing the tree, holding a mouse button down allows the tree to be dragged anywhere in the Browser window and the mouse wheel can be used to zoom in and out.

When reading the concept tree, the normal rules of attribute and object inheritance apply: attributes are inherited down the branches of the tree, objects are inherited up the branches. Thus, again looking at concept 2 in Figure 4, the objects that need water to live and live on land are frog, dog, corn, reed and bean. Looking at concept 8, frog and dog can move, have limbs, live on land and need water to live.

Figure 5 shows the Live-in-water concept tree fully expanded (a result most quickly achieved by selecting the 'Fully Expanded' option before uploading the tree).

When reading a formal concept tree it is important to realize that attributes and objects can be labeled more than once, something not present in a lattice where the the requisite 'many-to-many' connections ensure they are labeled only once. Nevertheless, the concepts in a concept tree are exactly the same concepts as those in the corresponding concept lattice.

The tree is best read from top to bottom in a manner of attribute exploration, or specialization, of concepts. Thus, in Figure 5, concept 0 contains all the objects that need water to live, concept 4 contains all the objects that need water to live *and* can move, and concept 5 contains all the objects that need water to live *and* can move *and* have limbs. However, concept 5 does not necessarily tell

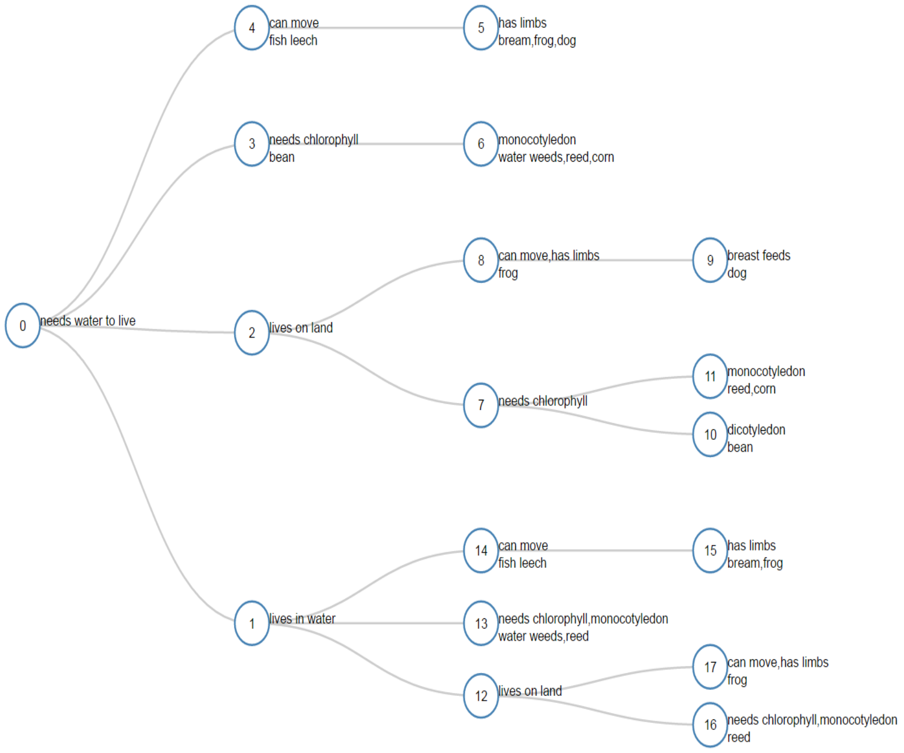

0 needs water to live
4 can move fish leech
5 has limbs bream,frog,dog
3 needs chlorophyll bean
6 monocotyledon water weeds,reed,corn
2 lives on land
8 can move,has limbs frog
9 breast feeds dog
7 needs chlorophyll
11 monocotyledon reed,corn
10 dicotyledon bean
1 lives in water
14 can move fish leech
15 has limbs bream,frog
13 needs chlorophyll,monocotyledon water weeds,reed
12 lives on land
17 can move,has limbs frog
16 needs chlorophyll,monocotyledon reed

Fig. 5. Fully Expanded Live-in-water Concept Tree

you about *other* attributes its objects may have. For example, a frog can also live in water (concept 1) and a dog can also live on land (concept 2). Although this issue is also present when reading a lattice, the extra connections in the lattice allow a user to determine those other attributes more easily. But, so long as this limitation is kept in mind, the concept tree is still a useful and readable visualisation.

4 'Iceberg Trees'

The idea of a formal concept 'iceberg' lattice was described by [12] as being a lattice where a minimum support has been applied, in terms of a minimum number of objects allowed in a concept. The original lattice is thus pruned of some of its lower concepts, reducing its size and complexity. However, the result of applying such minimum support to a lattice does not, strictly speaking, produce a lattice: the top portion remains as a lattice structure, but where it has been pruned there may be 'hanging branches', no longer connected by the pruned concepts. The same problem is not true of the 'iceberg tree': applying minimum support to a concept tree will simply produce a smaller tree. Such 'iceberg trees' are easily created using In-Close2.8 by specifying a minimum support before concept mining is carried out. Figure 6 shows the Live-in-water example with a minimum support of three objects - any concept with fewer than three objects has been pruned from the original Live-in-water tree.

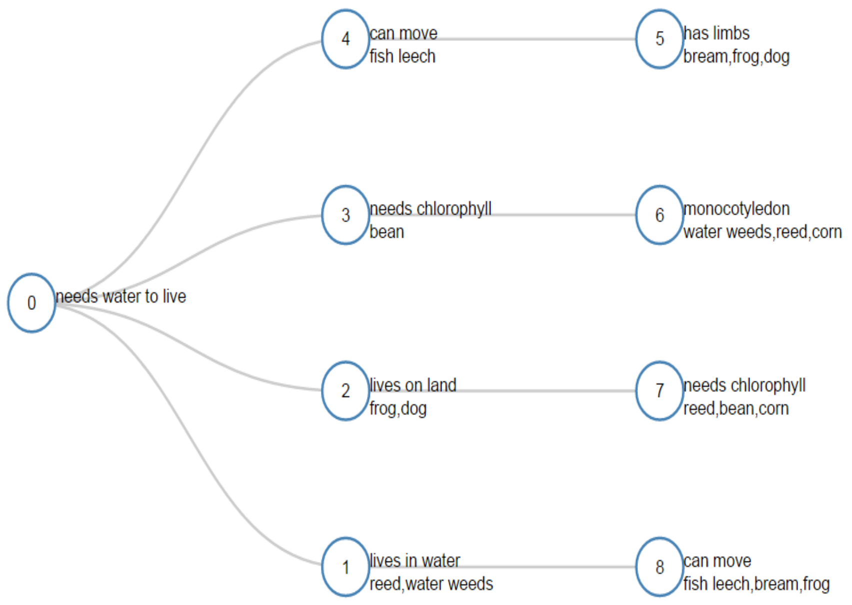


Fig. 6. Iceberg Live-in-water Tree with Min Supp of Three Objects

Taking concept 2, for example, concepts 8 and 9 from the full tree in Figure 5 have gone, and their objects (frog and dog) are now labeled at concept 2. It is important that such objects are not lost by the pruning and In-Close takes care of this by retaining objects as a concept's 'own objects' if they otherwise would belong to sub-concepts that don't satisfy the minimum support.

5 Managing Large Trees

Although pruning using minimum support will reduce the size of a tree, information, of course, will be lost. However, every tree, no matter how large, begins with a single node, and, so, with expandability and collapsibility, it becomes possible to manage very large trees, expanding and exploring branches of interest whilst leaving the rest of the tree hidden. Figure 7 shows a small portion of a concept tree generated from the well known 'mushroom' data set from UCI [7]. Altogether the tree has over 225,000 nodes and, although this would

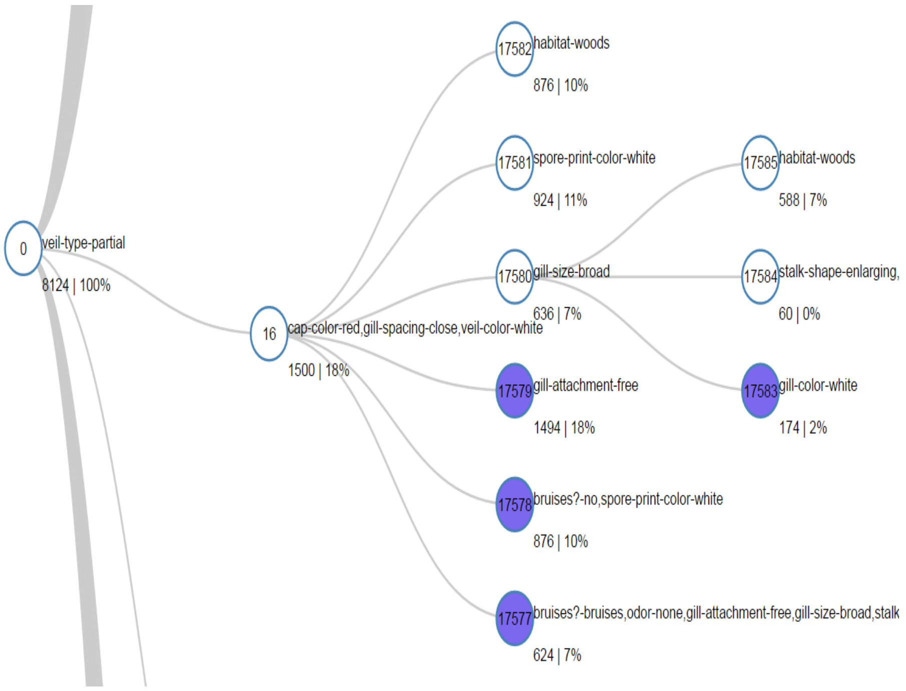

0 veil-type-partial
8124 | 100%
16 cap-color-red,gill-spacing-close,veil-color-white
1500 | 18%
17582 habitat-woods
876 | 10%
17581 spore-print-color-white
924 | 11%
17580 gill-size-broad
636 | 7%
17579 gill-attachment-free
1494 | 18%
17578 bruises?-no,spore-print-color-white
876 | 10%
17577 bruises?-bruises,odor-none,gill-attachment-free,gill-size-broad,stalk
624 | 7%
17585 habitat-woods
588 | 7%
17584 stalk-shape-enlarging,
60 | 0%
17583 gill-color-white
174 | 2%

Fig. 7. Portion of the 'mushroom' Concept Tree

clearly not be sensible to display in its entirety, the JavaScript program is able to store and manage the entire tree.

Notice that instead of displaying individual objects, the option to show 'object count' has been used, displaying the number of objects in a concept and the corresponding percentage of the population. The objects in this example are simply instances in a data set, so displaying long lists of their ID numbers makes little sense. The analyst is more interested in the statistical evidence of how attributes are distributed.

6 Conclusion

The concept tree tool is an addition to the FCA library of tools and provides a new automated approach to visualisation in FCA. Although the concept lattice is the natural visualisation of a concept hierarchy, the tree is a useful alternative that has not previously been provided in a tool. Trees are easily created from formal contexts using the In-Close2 concept miner and easily visualised in a Web Browser using the D3.js Collapsible Tree layout. The tool is able to manage trees with hundreds of thousands of nodes since size and complexity is dealt with by the expandable and collapsible nature of the layout and by the ability to prune trees by specifying a minimum support for objects in In-Close2.

The modified In-Close2 program is available at SourceForge [2] and the JavaScript program to visualise the tree in a Web-Browser can be accessed via the Web [3].

References

1. S. Andrews. In-close2, a high performance formal concept miner. In S. Andrews, S. Polovina, R. Hill, and B. Akhgar, editors, *Conceptual Structures for Discovering Knowledge - Proceedings of the 19th International Conference on Conceptual Structures (ICCS)*, pages 50–62. Springer, 2011.
2. S. Andrews. A partial-closure canonicity test to increase the efficiency of cbo-type algorithms. In *Proceedings of the 21st International Conference on Conceptual Structures*, pages 37–50. Springer, 2014.
3. S. Andrews. A best-of-breed approach for designing a fast algorithm for computing fixpoints of galois connections. *Information Sciences*, 295:633–649, 2015.
4. P. Becker and J.H. Correia. *The ToscanaJ Suite for Implementing Conceptual Information Systems*, volume 3626 of *LNCS*, pages 324–348. Springer, 2005.
5. R. Belohlavek, B. De Baets, J. Outrata, and V. Vychodil. Inducing decision trees via concept lattices. *International journal of general systems*, 38(4):455–467, 2009.
6. P.V. Borza, O. Sabou, and C. Săcărea. Openfca, an open source formal concept analysis toolbox. In *Automation Quality and Testing Robotics (AQTR)*, volume 3, pages 1–5. IEEE, 2010.
7. A. Frank and A. Asuncion. UCI machine learning repository: http://archive.ics.uci.edu/ml, 2010.
8. P. Krajca, J. Outrata, and V. Vychodil. Parallel recursive algorithm for FCA. In R. Belohavlek and S.O. Kuznetsov, editors, *Proceedings of Concept Lattices and their Applications*, 2008.
9. O. Kuznetsov, S. A fast algorithm for computing all intersections of objects in a finite semi-lattice. *Automatic Documentation and Mathematical Linguistics*, 27(5):11–21, 1993.
10. C. Melo, B. Le-Grand, M-A. Aufaure, and A. Bezerianos. Extracting and visualising tree-like structures from concept lattices. In *Proceedings of the15th International Conference on Information Visualisation*, pages 261–266. IEEE, 2011.
11. U. Priss. Fcastone-fca file format conversion and interoperability software. *Conceptual Structures Tools and the Web*, page 33, 2008.
12. G. Stumme, R. Taouil, Y. Bastide, and L. Lakhal. Conceptual clustering with iceberg concept lattices. *Proc. GI-Fachgruppentreffen Maschinelles Lernen (FGML01)*, 2001.
13. P. Valtchev, D. Grosser, C. Roume, and M. R. Hacene. Galicia: an open platform for lattices. In *A. de Moor B. Ganter, editor, Using Conceptual Structures: Contributions to 11th Intl. Conference on Conceptual Structures*, pages 241–254, 2003.
14. P. Velardi, R. Navigli, A. Cuchiarelli, and R. Neri. Evaluation of ontolearn, a methodology for automatic learning of domain ontologies. *Ontology Learning from Text: Methods, evaluation and applications*, 123:92, 2005.
15. S. A. Yevtushenko. System of data analysis ”concept explorer”. (in russian). In *Proceedings of the 7th national conference on Artificial Intelligence KII-2000*, pages 127–134, 2000.

[2] `https://sourceforge.net/projects/inclose/`

[3] `http://homepages.shu.ac.uk/~aceslh/fca/fcaTree.html`