Evaluation of Different BDD Libraries to Extract Concepts in FCA – Perspectives and Limitations

Andrei Rimsa, Luis E. Zárate, and Mark A.J. Song

Department of Computer Science, Applied Computational Intelligence Laboratory
Pontifical Catholic University of Minas Gerais - Brazil
rimsa@live.com, {zarate,song}@pucminas.br

Abstract. This paper presents evaluation of different types of Binary Decision Diagrams (BDDs) applied to Formal Concept Analysis (FCA). The aim is to increase the FCA capability to handle large formal contexts and perform faster operations over different types of this data structure. The main idea is to represent formal context using BDDs for later extraction of the set of all formal concepts from this implicit representation. A comparison of a concept extraction algorithm using contexts implemented as table and BDD are presented. BDD is evaluated over two different implementation libraries, BuDDy and CUDD. A ZBDDs (Zero-Supressed BDDs) version of the concepts extraction algorithm is also provided. BDD has been evaluated based on several types of randomly generated synthetic contexts with large amounts of objects. Contexts are evaluated according to the computational time complexity required to build and extract the set of all concepts from it. In this work, it is shown that BDD could be used to deal with large formal contexts especially when those have few attributes and many objects. To overcome the limitations of having contexts with fewer attributes, one could consider vertical partitions of the context to be used with distributed FCA algorithms based on BDD.

Keywords: Formal Concept Analysis, Formal Context, Formal Concept, Binary Decision Diagrams, Zero-Supressed Binary Decision Diagrams.

1 Introduction

At the International Conference on Formal Concept Analysis in Dresden (ICFCA 2006) an open problem of "Handling large contexts" was pointed out and as an example was cited the challenge of "how to calculate/generate all concepts of a large context" (e.g. 120,000 x 70,000 objects attributes). In these cases, traditional FCA algorithms have high computational cost and demand high execution times, making the extraction of all concepts infeasible for larger contexts.

One possible solution to deal with the problem of handling large formal contexts is to apply a distributed solution for the processing of contexts. Partial concepts are obtained for later merging through a specific operator to find the final set of concepts. Several authors have presented formal proposals and mathematical formalisms for distributed application of FCA, as can be seen in [1-3].

It is clear the potential of FCA to represent and extract knowledge from a set of objects and attributes and it is even more clear the problem of dealing with databases

G. Allen et al. (Eds.): ICCS 2009, Part I, LNCS 5544, pp. 367–376, 2009.

of high dimensionality. Application in real problems often suffers from this common fact. In this work, an approach to meet the challenge mentioned above consists in applying Binary Decision Diagrams (BDDs) [4] to obtain a symbolic representation of a cross table (formal context) that allows a more efficient extraction of the set of all concepts. It will be shown that this approach is promising and that it can handle more efficiently with large contexts when compared with the conventional implementation of algorithms that handle standard tables.

Although BDD has already been used in the FCA to represent the concept lattice [5], this article focus in the representation of formal contexts in BDD to achieve faster concepts extraction. Different BDD libraries will be used to evaluate BDD, including the BuDDy [6] and CUDD [7] package. Both support several variable reordering methods and present a C++ interface that reference nodes automatically and dereference them accordingly by the garbage collector. In addition, CUDD has built-in Zero-Suppressed BDDs [8] capabilities that were also evaluated in this work.

This article is organized in five sections. In the second section, the main concepts of the FCA and BDD are reviewed. In the third section, examining the representation of formal contexts through BDD and the extraction of concepts from this implicit representation is discussed. In the fourth section, BDD is evaluated over several large formal contexts. In the last section, the conclusions and future works are pointed out.

2 Formal Context

2.1 Formal Concept Analysis

Formal Context. Formal contexts have the notation $K:=(G, M, I)$, where G is a set of objects (rows headers), M is a set of attributes (columns headers) and I is an incidence relation ($I \subseteq G \times M$). If an object $g \in G$ and an attribute $m \in M$ are in the relation I, it is represented by $(g, m) \in I$ or gIm and is read as "*the object g has the attribute m*".

Given a set of objects $A \subseteq G$ from a formal context $K:=(G, M, I)$, it could be asked which attributes from M are common to all those objects. Similarly, it could be asked, for a set $B \subseteq M$, which objects have the attributes from B. These questions define the derivation operators, which are formally defined as:

$$A' := \{m \in M \mid gIm \; \forall \; g \in A\} \; ; \; B' := \{g \in G \mid gIm \; \forall \; m \in B\} \tag{1}$$

A special case of derivate sets occurs when empty sets of objects or attribute are considered to be derivate:

$$A \subseteq G = \emptyset \Rightarrow A' := M \; ; \; B \subseteq M = \emptyset \Rightarrow B' := G \tag{2}$$

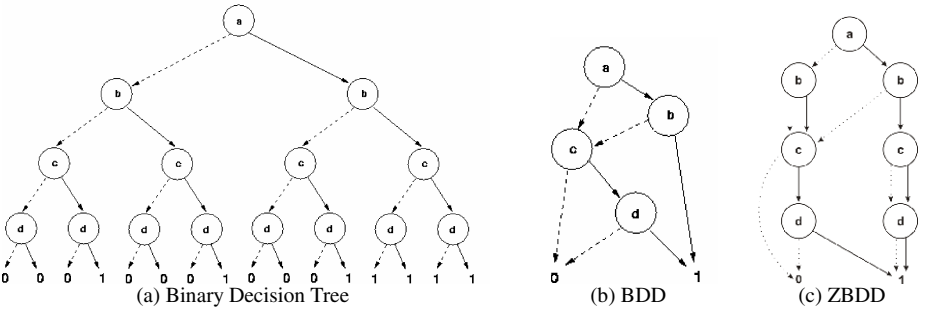
Formal Concept. Formal concepts are pairs (A, B), where $A \subseteq G$ (called extent) and $B \subseteq M$ (called intent). Each element of the extent (object) has all the elements of the intent (attributes) and, consequently, each element of the intent is an attribute of all objects of the extent. The set of all formal concepts in a formal context has the notation $\underline{\mathcal{B}}(G, M, I)$. Since that a cross table representing a formal context is given, algorithms can be applied in order to determine its formal concepts and its lattice [9].

2.2 Binary Decision Diagrams

Binary decision diagrams are a canonical representation of boolean formulas [4]. The BDD is obtained from a binary decision tree by merging identical subtrees and eliminating nodes with identical left and right siblings. The resulting structure is a graph rather than a tree in which nodes are eliminated and substructures are shared.

Formally, a BDD is a directed acyclic graph with two types of vertex: non-terminal and terminal. Each non-terminal vertex is a distinct variable of the corresponding boolean formula. Also, each vertex has two outgoing arcs directed toward two children, corresponding to the case where variable is 0 *(left)* and 1 *(right)*. A BDD has two terminal vertices labeled by 0 and 1, representing the truth-value of the formula false and true, respectively. For every truth assignment to the boolean variables of the formula, there is a corresponding path from root to a terminal vertex.

Zero-Suppressed BDD (ZBDD) is a graph representation similar to a BDD. ZBDD also represents boolean formulas by elimination nodes and sharing subtrees. However, as opposed to BDD, it does not eliminate nodes whose two edges, left and right arcs, points to the same node. It presented another elimination rule in which all nodes that the right arc points to the zero terminal node are removed [8]. The BDD rule to merge identical subtrees is still present in ZBDD.



(a) Binary Decision Tree (b) BDD (c) ZBDD

Fig. 1. Graph representation for formula (a ∧ b) ∨ (c ∧ d)

Figure 1 illustrates a BDD and a ZBDD compared to a Binary Decision Tree for the boolean formula *(a ∧ b) ∨ (c ∧ d)*. Note in the ZBDD diagram that the nodes with two arcs pointing to the same node weren't removed like it would be with BDD. But nodes in ZBDD can still be removed when the new elimination rule is applied.

BDDs are an efficient way to represent boolean formulas. Often, they provide a much more concise representation compared to the traditional representations, such as conjunctive and disjunctive normal forms. BDDs are also a canonical representation for boolean formulas. This means that two boolean formulas are logically equivalent if and only if its BDDs are isomorphic. This property simplifies the execution of frequent operations, like checking the equivalence of two formulas.

However, BDD has drawbacks. The most significant is related to the order in which variables appear. Given a boolean formula, the size of the corresponding BDD is highly dependent on the ordering. It can grow from linear to exponential according to the number of variables of the formula. In addition, the problem of choosing a

variable order that minimize the BDD size is NP-complete [4]. Despite the existence of heuristics to automatically order the variables, they are often ordered manually.

3 Formal Concepts Extraction Using BDD

When formal contexts are represented as BDDs, it is possible to implement derivation operators to work directly over this representation, thus allowing FCA algorithm independence. Unfortunately, the cost to identify the set of objects from a BDD concept is too expensive, thus invalidating this alternative. To overcome this problem, algorithms to extract concepts and/or to construct the concept lattice available in the literature must be adapted to handle this new form of representation.

To demonstrate the feasibility of BDD, the adapted algorithm was the Attribute Intersections [10] because of its inherent characteristics that allow a more effectively concepts extraction from contexts with more objects than attributes. This algorithm implementation in BDD was divided in three primary stages (Fig. 2). In the first stage, the construction of the context in BDD is made. The second stage is responsible to extract the set of all concepts from the BDD context. The final stage is responsible to identify the attributes and objects from the concepts represented in BDD. These stages separation avoids unnecessary high cost operations while obtaining the concepts.

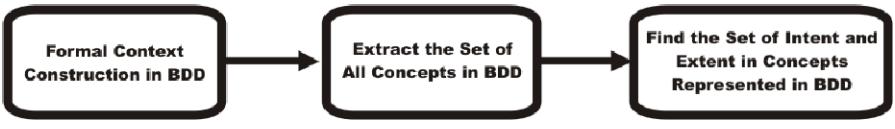


Fig. 2. Steps to implement the Attribute Intersection algorithm in BDD

3.1 Formal Context Construction in BDD

To create the BDD representation, a formal context must be converted into an equivalent logic boolean formula. Table 1 shows an example of a formal context and its possible representation through a logic function (Equation 3).

Table 1. Formal Context Example

	a1	a2	a3
o1	**X**		**X**
o2	**X**	**X**	
o3		**X**	

$$f(a_1, a_2, a_3) = a_1\overline{a_2}a_3 + a_1a_2\overline{a_3} + \overline{a_1}a_2\overline{a_3} \quad (3)$$

Note that each object is represented by a logic equation, according to the presence or not of its attributes. The function $f(a_1, a_2, a_3)$ results in a positive state (1) when an object is present on the context. This function returns the negative state (0) for objects not present in the context. Thus, any context can be represented by a logic function.

Algorithm 1 allows the construction of BDD based on the objects presented in the formal context. The internal functions *bdd_ithvar* and *bdd_nithvar* are specific to the library BuDDy [6] and are used to define the presence or not of an attribute in the BDD, respectively. The CUDD library has it own function wrappers to perform this

operation, *Cudd_bddIthvar*. To obtain the respective negative variable, CUDD requires the use of *Cudd_Not* in conjunction with the *Cudd_bddIthvar*. Once the conjunction of attributes is made forming the objects (lines 7 and 9) then a disjunction of those objects is realized (line 12) to build the context.

Algorithm 1. BDD construction based on the context.

```
in:  List<Object> list
out: BDD context
 1: context = bddfalse
 2: while !list.empty( ) do
 3:   obj = list.removeFirstObject( );
 4:   BDD tmp = bddtrue
 5:   for i=0; i<obj.attributes; i++ do
 6:     if obj.hasAttribute(i) then
 7:       tmp &= bdd_ithvar(i)
 8:     else
 9:       tmp &= bdd_nithvar(i)
10:     endif
11:   done
12:   context |= tmp
13: done
```

Normally, ZBDDs are constructed by converting from a previously created BDD. However, to maintain coherence with the Algorithm 1, in this work, the ZDD was built object by object. CUDD maintains a different structure to operate ZBDD and positive variables reference may be achieved using the *Cudd_zddIthVar*. Since ZBDDs works with one's complement, it is not possible to use the standard *Cudd_Not* approach to obtain the negative part of a variable. But it can be obtained by the *Cudd_zddDiff* function, using the resulting difference from a 1-node constant, thus acquiring the complement.

In this work, references and dereferences of nodes were made manually when the CUDD library was used. In the other hand, the BuDDy C++ wrapper was used that allowed automatic references and dereferences of nodes. The garbage collector dereferences nodes automatically when memory resources are claimed.

It is important to emphasize that the main objective of this work is to show the feasibility of BDD to represent formal contexts, and from that representation extract the formal concepts. The feasibility is shown through the manipulation of large formal contexts. In most cases, the BDD representation of contexts often consumes several memory resources. However, it is not significant enough to invalidate this new representation in BDD. So the BDD can be used to extract concepts more efficiently than the algorithms that work directly in the tabular representation.

3.2 Extracting the Set of All Concepts in BDD

Algorithm 2 is the kernel of the Attribute Intersection algorithm, but slightly modified to work with BDD. This implementation in BDD takes advantage of two distinct moments when the derivation operator is used (Line 4) and the intersection between two concepts is made (Line 8). The derivation operator is easily implemented through the implicit *bdd_ithvar* operator, which obtains a BDD representation of all objects

with an attribute. The intersection between two concepts is also implemented through an implicit BDD operation, *bdd_and* (&). Moreover, the concepts list was implemented as a *hashtable* to achieve a faster verification of concepts duplicity. The algorithm kernel is the same for the CUDD version with BDD and ZBDD.

Algorithm 2. BDD construction based on the context.

```
in:  BDD context
out: List<BDD> concepts
 1: concepts = new List<BDD>
 2: concepts.addConcept(context)
 3: for i=0; i<attributes; i++ do
 4:   BDD tmp1 = context & bdd_ithvar(i)
 5:   size = concepts.size()
 6:   for j=0; j<size; j++ do
 7:     BDD tmp2 = concepts.getConcept(j)
 8:     BDD intersection = tmp1 & tmp2
 9:     if !concepts.exist(intersection) then
10:       concepts.add(intersection)
11:     endif
12:   done
13: done
```

Unfortunately, storing all the concepts as BDD in the list reflects a very expressive memory consumption. The algorithm was slightly modified to save the concept intent (B_i) rather than the concept (A_i, B_i) in BDD. From the intent set (B_i), one can rebuild the concept in BDD, thereby maintaining the essence of the proposed Algorithm 2.

3.3 Finding the Set of Intent and Extent in Concepts Represented in BDD

This section shows how to obtain the extent and intent of the concepts represented in BDD. Algorithm 3 is used to check if all objects represented by the BDD share an attribute in common. Algorithm 4 is used to verify whenever an object is present in the BDD concept.

For the extraction of all objects (extent) of the concept, Algorithm 4 can be used to verify if each object that exists in the formal context is present in the concept. The same can be applied to the set of attributes (intent), through Algorithm 3, covering all formal context attributes checking whether or not they are present in the concept. Also, in Algorithm 4, the BuDDy *bdd_varlevel* operation has no correspondence in the CUDD library, but can be obtained by their node index value.

Algorithm 3. Verify the presence of an attribute in a concept represented in BDD.

```
in:  BDD concept, attr
out: presence
 1: BDD tmp = concept & bdd_ithvar(attr)
 2: if tmp == concept then
 3:   present = true
 4: else
 5:   present = false
 6: endif
```

Algorithm 4. Verify the presence of an object in a concept represented in BDD.

```
in:  BDD concept, objc
out: presence
 1: BDD tmp = concept
 2: i = 0
 3: while i<objc.attributes and
 4:       tmp != {bddtrue, bddfalse} do
 5:   if bdd_varlevel(tmp) == i then
 6:     if obj.hasAttribute(i) then
 7:       tmp = bdd_high(tmp)
 8:     else
 9:       tmp = bdd_low(tmp)
10:     endif
11:   endif
12:   i++
13: done
14: presence =(tmp == bddtrue)
```

Although Algorithm 4 is used to identify if an object exists in a BDD concept, it was not used in the Attribute Intersection implementation. As mentioned, only the concepts intents are store in the list. With the intents it is not necessary to rebuild the BDD concept to verify objects presence. It can be done directly by checking if every object has the attributes of each of the intent sets on the list.

4 Feasibility Analysis of BDD to Extract Concepts

One of the requirements to assess the representativeness of BDD to extract concepts was to compare its performance under the same conditions as its tabular version. For this reason, it was decided to implement a unique algorithm for the situations: contexts represented in BDD (with BuDDy and CUDD), ZBDD (with CUDD) and by a table. Also, several optimizations on the table version were made to meet this purpose. Moreover, SCGaz (available at http://www.inf.pucminas.br/projetos/licap) was the tool responsible to build all contexts used in this work. This tool allows the construction of semi-clarified contexts, avoiding some types of attributes and objects redundancy, like repeated objects and also empty and full attributes and objects.

Figure 3 shows the behavior of the Attribute Intersections algorithm for the BDD with BuDDy and CUDD, ZBDD and tabular version for contexts with fewer attributes (20 to 100) and many objects (10,000 to 60,000). To ensure that the BDD would not be extremely compact, the used density for all contexts was the minimum plus 10% of it. Moreover, lower density values result into smaller amounts of concepts, thus making the simulations consumes less time to execute. All software were written in C++ and executed on a Pentium Dual Core 2.66GHz with 2Gb of RAM running Linux Slackware 12.0.

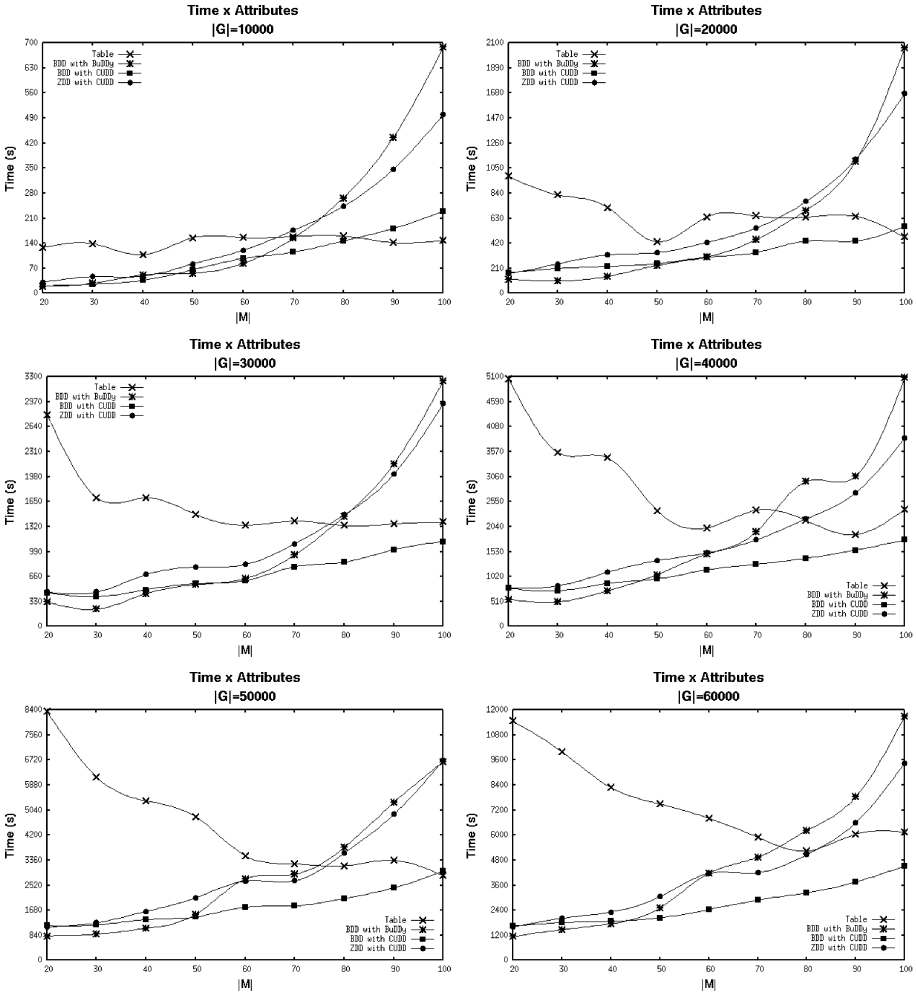


Fig. 3. Evaluation of Attribute Intersections implemented as a table, BDD and ZBDD

The tabular version has presented an irregular decreasing behavior because of the density, since with few attributes higher will be the density for these considered simulations. In addition to that, how the incidences are spread into the context can explain its irregular behavior. The BDDs and ZBDD had exponential behavior, even the CUDD version that may appear linear. Increase the attributes may allow to observe its exponential behavior. With more attributes, more nodes will be required to construct the BDD. Therefore, less efficient will be the operations in this representation. Also, as can be seen by simulations of 20 and 30 attributes, while the tabular version had worst time performance, the BDD maintained a very low execution time, despite of the higher density. So, the context BDD size is extremely relevant to the extraction of all concepts.

Table 2. Execution time for many-valued context with |M|=70, |G|=60,000 and |g'|=7

	Construction of the Context (s)	Concepts Extraction (s)	Intent and Extent Identification (s)	Total (s)	Total
Table	-	-	-	81059	0d 22:30:57
BuDDy BDD	51	10151	9946	20148	0d 05:35:48
CUDD BDD	192	14075	10107	24374	0d 06:46:14
CUDD ZBDD	306	10272	9703	20281	0d 05:38:01

Table 3. Execution time for many-valued context with |M|=70, |G|=120,000 and |g'|=7

	Construction of the Context (s)	Concepts Extraction (s)	Intent and Extent Identification (s)	Total (s)	Total
Table	-	-	-	251283	2d 21:48:03
BuDDy BDD	128	18345	33289	51762	0d 14:22:42
CUDD BDD	569	36841	33043	70453	0d 19:34:13
CUDD ZBDD	629	22107	34844	57580	0d 15:59:40

The BDD with CUDD package had better performance over the table in all simulations, making it strongly reliable. It would be necessary more simulations with higher attributes to verify if there is a threshold in which the table begins to be more worthed. BDD with BuDDy and ZBDD had better performance over table when the number of concepts is not superior to 70. As the amount of objects increases, greatest has become the difference between the execution times of BDD implementations compared to the table. Thus, the implicit representation of concepts in BDD becomes an alternative to a more efficient extraction of concepts in these conditions.

Considering now a threshold of 70 attributes, another simulation scenario was created. A many-valued context was simulated with 7 attributes, a fixed number of 10 attribute-values per attribute and in two situations with objects, 60,000 and 120,000. This type of context has 7 attributes per object (|g'|=7). Table 2 and 3 presents the spent time consumed by these situations.

For this type of context, BDD and ZBDD had outstanding improvement over the tabular version. As opposed to the previous simulations, the BDD with CUDD presented the least performance. In spare context, ZBDD was proven faster, but not enough to beat BuDDy performance. Also, CUDD required more time to build the context, possibly affecting the final BDD size. BDD with BuDDy and ZBDD had similar execution times for contexts with 60,000 objects, but taking the ICFCA'06 challenge with 120,000 objects, the difference was quite significant, over 1 hour and a half. Compared to the table version, the difference between the BuDDy version was almost three days. Applicability to process larger contexts could be achieved with the use of a distributed version of an algorithm implemented in BDD.

Note that the required time to identify the set of extents from the concepts represented in BDD was very significant, as seen by Table 2 and 3. This happens because of the used algorithm quadratic complexity relative to the number of objects. If more efficient algorithms were used, lower computational times may be achieved to process contexts. Instead of a brute force strategy to check objects presence in a concept, another strategy could be visiting BDD nodes in order to identify the objects.

5 Conclusions

The present work is related to a challenge raised at the ICFCA'06 conference, which refers to the manipulation of large formal contexts. Through the use of an implicit representation of formal context in BDD, it has been demonstrated that this new representation became computationally feasible for handling large contexts, when compared to the conventional manipulation of a table. Although the BDD allows the manipulation of contexts with a large number of objects, it is restricted to contexts with few attributes. Thus, if the context meets this feature, a significant efficiency can be achieved with the application of this new alternative.

As future work, more robust FCA algorithms could be adapted to use BDD or ZBDD. It's essential to verify the feasibility of BDD application in a fast algorithm, like CHARM [11]. Even faster concepts extraction may be achieved with others algorithms applied with BDD.

References

1. Li, Y., Liu, Z.T., Shen, X.J., Wu, Q., Qiang, Y.: Theoretical research on the distributed construction of concept lattices. In: International Conference on Machine Learning and Cybernetics, vol. 1, pp. 474–479 (2003)
2. Liu, Z., Li, L., Zhang, Q.: Research on a union algorithm of multiple concept lattices. In: Wang, G., Liu, Q., Yao, Y., Skowron, A. (eds.) RSFDGrC 2003. LNCS (LNAI), vol. 2639, pp. 533–540. Springer, Heidelberg (2003)
3. Lévy, G., Baklouti, F.: A distributed version of the ganter algorithm for general galois lattices (2005)
4. Bryant, R.: Graph-based algorithms for boolean function manipulation. IEEE Transactions on Computers C-35(8), 677–691 (1986)
5. Yevtushenko, S.: Computing and visualizing concept lattices. PhD thesis, TU Darmstadt, Fachbereich Informatik (2004)
6. Lind-Nielsen, J.: Buddy: A binary decision diagram. Technical report, Department of Information Technology, Technical University of Denmark, Lyngby, Denmark (1999), `http://www.itu.dk/research/buddy`
7. Somenzi, F.: CUDD: CU decision diagram package release (1998)
8. Minato, S.: Zero-suppressed BDDs for set manipulation in combinatorial problems. In: DAC 1993: Proceedings of the 30th International Conference on Design Automation, pp. 272–277. ACM, New York (1993)
9. Grätzer, G.: General Lattice Theory. Birkhäuser, Basel (1978)
10. Carpineto, C., Romano, G.: Concept Data Analysis: Theory and Applications. John Wiley & Sons, Indianapolis (2004)
11. Zaki, M., Hiao, C.: ChARM: An efficient algorithm for closed association rule mining. Technical Report 99-10, Computer Science Dept., Rensselaer Polytechnic Inst., Troy, NY, USA (October 1999)