Handling Large Formal Context Using BDD – Perspectives and Limitations

Andrei Rimsa, Luis E. Zárate, 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 Binary Decision Diagrams (BDDs) applied to Formal Concept Analysis (FCA). The aim is to increase the FCA capability to handle large formal contexts. The main idea is to represent formal context using BDDs for later extraction of the set of all formal concepts from this implicit representation. BDDs have been evaluated based on several types of randomly generated synthetic contexts and on density variations in two distinct occasions: (1) computational resources required to build the formal contexts in BDD; and (2) to extract all concepts from it. Although BDDs have had fewer advantages in terms of memory consumption for representing formal contexts, it has true potential when all concepts are extracted. In this work, it is shown that BDDs 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 BDDs.

Keywords: Formal Concept Analysis, Formal Context, Formal Concept, 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]. However, these contributions do not analyze the performance aspects of the distributed version concerning the density impact on the context.

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 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 [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 handles tables. Although BDD suffers from limitations of handling contexts with many attributes, a common problem faced by FCA, it can handle huge amount of objects, making it thus reliable for some set of problems. Also, in these conditions, the BDD representation presents computational improvements. In some cases it can even save days of processing, as it will be later addressed.

BDD has already been used earlier in FCA. In [5], using previously obtained concepts, a concept lattice is built based on ZBDDs (Zero-Suppressed BDDs) [6], a type of BDD. In this paper, BDD have been applied with a different aim, to represent formal contexts in order to improve concepts computation. This article presents an analysis of this new representation, both in its impact in memory consumption as the computational time required to execute an algorithm to extract all concepts.

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 is discussed. In the fourth section, the principles and algorithm for extraction of formal concepts from BDD are presented. 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:

$$A' := \{m \in M \mid gIm \ \forall g \in A\} \qquad (1)$$

$$B' := \{g \in G \mid gIm \ \forall m \in B\} \qquad (2)$$

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

$$A \subseteq G = \emptyset \Rightarrow A' := M \; ; \; B \subseteq M = \emptyset \Rightarrow B' := G \qquad (3)$$

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 $\underline{B}(G, M, I)$. From a cross table representing a formal context, algorithms can be applied in order to determine its formal concepts and its line diagram [7].

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 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 *v* is labeled by *var(v)*, a distinct variable of the corresponding boolean formula. Each *v* has at least one incident arc (except the root vertex). Also, each *v* has two outgoing arcs directed toward two children: *left(v)*, corresponding to the case where *var(v)=0*, and *right(v)* to the case where *var(v)=1*.

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 in the BDD from root to a terminal vertex. Figure 1 illustrates a BDD for the boolean formula $(a \wedge b) \vee (c \wedge d)$ compared to a Binary Decision Tree for this same formula.

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.

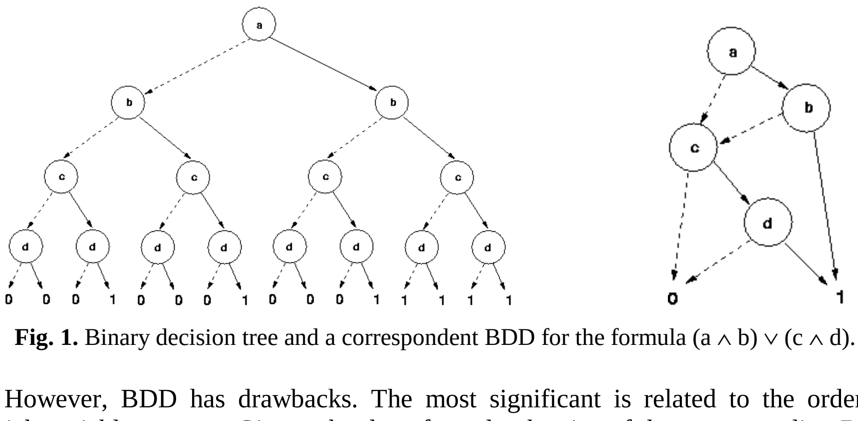


Fig. 1. Binary decision tree and a correspondent BDD for the formula $(a \wedge b) \vee (c \wedge d)$.

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 variable 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]. There are heuristics to order the BDD variables; some of them are based on the knowledge over the problem. A review of some heuristics can be found in [8].

3 Formal Contexts Represented in BDD

3.1 BDD Construction from Formal Contexts

As mentioned, BDD is able to represent logical expressions through simplified graphs. In this way, a context can be converted into an equivalent logic formula to be used in the creation of the BDD representation. Table 1 shows an example of a formal context and its possible representation through a logic function.

Table 1. Formal Context Example

	a_1	a_2	a_3
o_1	**X**		**X**
o_2	**X**	**X**	
o_3		**X**	

$$f(a_1, a_2, a_3) = a_1 \overline{a_2} a_3 + a_1 a_2 \overline{a_3} + \overline{a_1} a_2 \overline{a_3} \quad (4)$$

Note that each object is represented by a logic equation, according to the presence or not of its attributes. The function *f(a₁, a₂, a₃)* 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. BDD construction based on the context.

```
in:  List<Object> list
out: BDD context
context = bddfalse
while !list.empty( ) do
  obj = list.removeFirstObject( );
  BDD tmp = bddtrue
  for i=0; i<obj.attributes; i++ do
    if obj.hasAttribute(i) then
      tmp &= bdd_ithvar(i)
    else
      tmp &= bdd_nithvar(i)
    endif
  done
  context |= tmp
done
```

Algorithm 1 allows the construction of BDD based on the objects presented in the formal context. Note that this algorithm maintains the same attribute order of the formal context in order to build the context in BDD. The internal functions *bdd_ithvar* and *bdd_nithvar* are specific to the *BuDDy* [9] library and are used to define the presence or not of an attribute in the BDD, respectively. Once the conjunction of attributes is made, forming the objects (lines 7 and 9), then a disjunction of those objects is realized (line 12).

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 is less memory efficient when compared to a bit table representation of the contexts, as will be seen in the next section. But the memory consumption is not significant enough to invalidate its 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 BDD Representativeness Analysis

To achieve a more reliable simulation environment, all context used in this work were built by the SCGaz tool (available at http://www.inf.pucminas.br/projetos/licap) to evaluate the BDD performance. This tool can randomly generate formal contexts with user-defined densities while avoiding the existence of some type of redundant attributes and objects. The use of partially clarified contexts was considered in this work to guarantee that the final size of the BDD representation is not influenced by the existence of repeated objects and attributes, since this representation can internally simplify these redundancies. This is an important step to ensure fairness of the representation that could otherwise be used to mask the true performance of BDD. Note that the BDD representation is not restricted to clarified contexts and can be applied to any type of context, regardless of its information redundancy.

To construct and operate the BDD, the *BuDDy* library was used in which each node of its graph is represented by 20 bytes. Thus, the number of nodes in the graph was the parameter used to quantify the representation memory consumption. In order to compare the BDD representativeness, a relationship between the bit table memory consumption was stipulated. Where S_{table} and S_{bdd} correspond respectively to the table and BDD memory sizes. This metric calculates the gain (*Gain*) of a representation in relation to another. Thus, when the BDD consumes less memory than the bit table, equation (5) is then used. When the bit table is more efficient, the expression of equation (6) is therefore used. The negative sign indicates the loss of the BDD compared to the bit table memory consumption.

$$Gain = \left(\frac{S_{table}}{S_{bdd}} \right) \quad (5) \qquad Gain = -\left(\frac{S_{bdd}}{S_{table}} \right) \quad (6)$$

Note that the number of nodes in the BDD is not related to the number of filled cells of the context. Contexts with the same density and filled cells can present BDDs with more or less nodes.

To assess the representativeness of contexts through BDD, it is considered their behavior over different types of context: $|G|=|M|$, $|G|<|M|$, $|G|>|M|$ and many-valued. For each type of context considered, a pair (attribute, object) was simulated for 10 cases of density, ranging from its minimum to its maximum value. For the types of contexts with unique densities (many-valued and contexts $|G|>|M|$ with $|G|=2^{|M|\ -\ 2}$), a single simulation was performed. The implementations were made in C++ and the simulations were realized on a Pentium IV 3.06Ghz HT with 2GB of RAM running Slackware Linux 12.0.

Contexts |G| = |M|. Table 2 and Figure 2 correspond to the representation of contexts through BDD, where the number of objects is equal to the number of attributes. Table 2 shows the minimum, maximum, and the median values for the BDD memory consumption (S_{bdd}), as well as the required time to build the BDD (T_{bdd}) and the representativeness gain compared to the bit table. Figure 2 shows the gains obtained for the cases (100, 100), (1000, 1000) and (5000, 5000) in function of the density.

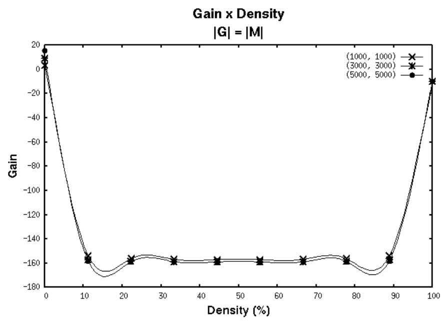


Fig. 2. BDD gain for contexts $|G|$ = $|M|$.

Table 2. BDD simulation for contexts $|G|$ = $|M|$.

|G|	|M|	S_{table} (Kb)	S_{bdd} (Kb)			T_{bdd} (s)			Gain		
			Min	Median	Max	Min	Median	Max	Min	Median	Max
50	50	0	2	36	39	0.01	0.02	0.03	-128.21	-117.18	-6.35
100	100	1	4	163	172	0.08	0.22	0.25	-140.64	-133.51	-3.18
300	300	11	12	1636	1669	3.27	9.06	9.18	-151.87	-148.94	-1.06
500	500	31	20	4660	4718	18.19	47.53	49.93	-154.60	-152.69	1.56
1000	1000	122	39	19038	19163	362.66	661.04	786.00	-156.98	-155.96	3.13
1500	1500	275	59	43157	43357	1217.48	2435.05	2585.38	-157.86	-157.13	4.69
2000	2000	488	78	77029	77310	3013.76	5959.37	6059.72	-158.33	-157.76	6.25
2500	2500	763	98	120662	121019	5728.60	11723.10	12051.70	-158.62	-158.15	7.81
3000	3000	1099	117	174050	174493	7252.94	14696.80	14829.50	-158.83	-158.42	9.38
3500	3500	1495	137	237203	237718	11450.90	23694.70	23907.90	-158.97	-158.63	10.94
4000	4000	1953	156	310110	310708	17159.50	35908.95	36225.80	-159.08	-158.78	12.50
4500	4500	2472	176	392771	393462	24577.30	52143.25	52561.80	-159.17	-158.89	14.06
5000	5000	3052	195	485201	485977	34321.40	73011.15	73679.80	-159.25	-158.99	15.63

As expected, BDD obtained a better performance in low and high densities (approximately below 10% and above 90%) when compared to intermediate densities. This occurs because with few or many incidences in the context table, there are several similarities in the BDD graph that allow simplifications of sub-expressions. The BDD is therefore represented with fewer nodes. Meanwhile, in the intermediate densities, the BDD had a constant behaviour. This happens because the BDD is using few objects compared to the total universe of objects ($2^{|M|}$), making no influence in the representation size. The BDD is unable to find enough objects similarities that could allow significant simplification. So, the intermediate density has no effect over the BDD representative size.

It is noteworthy that, although the BDD has had a lower performance than the bits table, the representation of formal contexts in BDD is computationally feasible. Through data collected by the simulations, it is pointed out that, for a context with 5,000 attributes and 5,000 objects, the BDD required expressively more memory than the bit table representation and was able to build it in a viable computational time of about 20 hours.

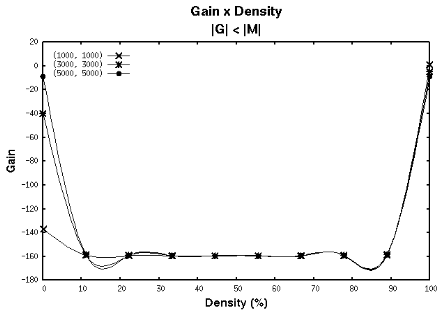


Fig. 3. BDD gain for contexts $|G| = |M|$.

Table 3. BDD simulation for contexts $|G| < |M|$.

|G|	|M|	S_{table} (Kb)	S_{bdd} (Kb)			T_{bdd} (s)			Gain		
			Min	Median	Max	Min	Median	Max	Min	Median	Max
100	1000	12	42	1921	1929	25.19	27.68	28.03	-158.05	-157.40	-3.43
600	1000	73	390	11444	11515	151.76	253.24	255.87	-157.22	-156.24	-5.33
900	1000	110	860	17140	17251	237.00	419.00	423.23	-157.02	-156.02	-7.83
200	2000	49	82	7739	7757	211.45	286.26	289.19	-158.87	-158.49	-1.67
1200	2000	293	1612	46263	46419	1318.07	2475.16	2498.55	-158.44	-157.91	-5.50
1800	2000	439	3664	69342	69589	2099.88	3783.11	3819.29	-158.35	-157.79	-8.34
300	3000	110	117	17457	17489	725.02	1267.96	1276.93	-159.19	-158.90	-1.06
1800	3000	659	3179	104497	104745	4678.94	8663.08	8734.99	-158.90	-158.53	-4.82
2700	3000	989	8648	156663	157054	9454.73	13168.50	13297.90	-158.84	-158.44	-8.75
400	4000	195	213	31079	31124	1725.56	3227.89	3258.88	-159.36	-159.12	-1.09
2400	4000	1172	4969	186152	186497	11886.00	20967.85	21207.00	-159.15	-158.85	-4.24
3600	4000	1758	15518	279119	279660	26804.10	32140.80	32479.40	-159.10	-158.79	-8.83
500	5000	305	273	48605	48664	3403.50	6507.70	6564.56	-159.46	-159.27	1.12
3000	5000	1831	9071	291238	291674	30512.60	42217.25	42495.30	-159.29	-159.06	-4.95
4500	5000	2747	23234	436726	437409	55495.90	64979.45	65884.40	-159.26	-159.01	-8.46

Contexts |G| < |M|. Table 3 presents the data collected in simulations to settings where the number of objects is less than the number of attributes. This is considered the worst situation for the BDD, since there is a small amount of objects, resulting in a low probability of finding objects with similar characteristics. Thus, the BDD is unable to simplify enough its representation to overcome the performance of the bits table. The representation of the bits table is still very compact in order to verify a representativeness gain of the BDD version. A similar behavior of Figure 2 can be observed in Figure 3. In extreme values of densities, the BDD performance had a slightly improvement over intermediate densities. However, the gains were not enough for the memory space consumed by the BDD representation suppress the bits table representation. Again, the bit table representation was extremely compact. The expressive number of attributes increases the BDD graph depth and requires more nodes to be represented.

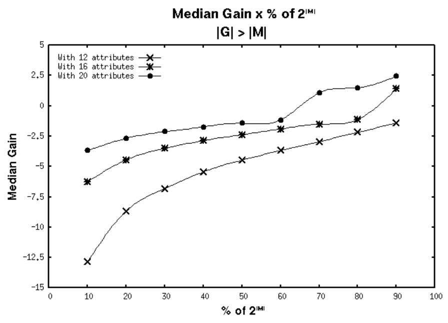


Fig. 4. BDD gain for contexts $|G| = |M|$.

Table 4. BDD simulation for contexts $|G| > |M|$.

|G|	|M|	S_{table} (Kb)	E_{bdd} (Kb)			T_{bdd} (s)			Gain		
			Min	Median	Max	Min	Median	Max	Min	Media	Max
102	10	0	1	2	3	0.00	0.00	0.00	-22.52	-20.08	-10.87
613	10	1	2	4	5	0.01	0.01	0.01	-6.16	-5.56	-2.27
920	10	1	1	3	3	0.01	0.01	0.01	-2.42	-2.23	-1.15
409	12	1	3	8	8	0.00	0.01	0.01	-14.03	-12.85	-5.55
2457	12	4	3	13	14	0.03	0.03	0.03	-3.88	-3.66	1.08
3685	12	5	4	8	8	0.04	0.04	0.04	-1.53	-1.41	1.53
1638	14	3	7	24	27	0.02	0.02	0.03	-9.57	-8.73	-2.46
9829	14	17	6	42	45	0.13	0.13	0.14	-2.66	-2.52	2.82
14744	14	25	7	24	27	0.19	0.20	0.21	-1.05	1.04	3.49
6553	16	13	13	80	88	0.10	0.12	0.13	-6.90	-6.25	-1.01
39321	16	77	10	145	160	0.77	0.78	0.92	-2.08	-1.89	7.90
58981	16	115	13	80	88	1.19	1.21	1.22	1.31	1.44	8.96
26214	18	58	66	271	298	0.61	0.71	0.77	-5.18	-4.70	-1.14
157285	18	346	55	515	582	4.70	4.97	5.01	-1.68	-1.49	6.25
235928	18	518	65	270	298	7.32	7.37	7.39	1.74	1.92	7.93
104857	20	256	248	938	1033	3.71	4.54	4.79	-4.03	-3.66	1.03
629145	20	1536	225	1759	1962	28.09	30.13	43.25	-1.28	-1.15	6.84
943717	20	2304	267	936	1034	39.58	40.26	40.33	2.23	2.46	8.62

Contexts |G| > |M|. The data shown in Table 4 reflects the simulations realized with a wider number of objects than attributes. For each type of context, the selected amount of objects was 10%, 60% and 90% of the maximum number of objects possible ($2^{|M|}$). It is noticed by Figure 4 that the gains and losses were not significant when a large amount of objects are used. The more the number of objects cover the total objects universe of possibilities ($2^{|M|}$), better will be the BDD representativeness. This is the case when 90% of the maximum number of objects is used. Also, BDD presented a stable behavior with fewer variations in the minimum, median and maximum gain for this type of context. With fewer attributes and many objects, BDD may become an attractive alternative to express data contained in cross tables. Also, the computational time required to assemble the BDD graph is not a limiting factor, allowing the construction of a BDD with 943,717 objects in times around 40 seconds. However, this situation is only reflected in context with few attributes. Increase the amount of attributes in the BDD has serious consequences in its size.

Many-Valued Contexts. Table 5 shows the data collected for many-valued contexts concerning attributes ranging from 1 to 5, where each attribute was simulated with 5, 10 and 15 intervals of discretization. All contexts used in these simulations have only one incidence per attribute, i.e. only one attribute-value by attribute. The amount of objects considered in the simulations is shown in Figure 5 along with the considered density. As it can be seen through Table 5, is possible to assemble a BDD context with more than 700,000 objects in approximately 13 minutes. The gains obtained in the data presented in Table 5 show that BDD has a satisfactory memory performance for this type of context. The density of this type of context is naturally lower, allowing the BDD representation to find more simplifications and be represented in a more compact form.

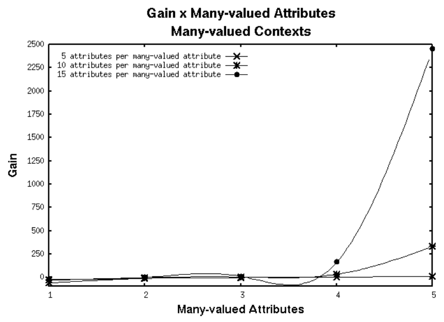


Fig. 5. BDD gain for many-valued contexts.

Table 5. BDD simulation for many-valued contexts.

$\|G\| = \|V\|^{\|M\|}$	$\|M\|$	$\|V\|$	$\|M\|\|V\|$	Den	S_{tabela} (Kb)	S_{bdd} (Kb)	T_{bdd} (s)	Gain
5	1	5	5	20%	0	0	0.00	-60.00
25	2	5	10	20%	0	0	0.00	-11.61
125	3	5	15	20%	0	0	0.00	-2.31
625	4	5	20	20%	1	0	0.01	2.17
3125	5	5	25	20%	10	0	0.08	10.85
10	1	10	10	10%	0	0	0.00	-31.67
100	2	10	20	10%	0	0	0.00	-3.04
1000	3	10	30	10%	4	1	0.04	3.29
10000	4	10	40	10%	49	1	0.72	32.89
100000	5	10	50	10%	610	2	24.86	328.95
15	1	15	15	6%	0	0	0.00	-20.71
225	2	15	30	6%	0	1	0.01	-1.38
3375	3	15	45	6%	19	2	0.26	10.91
50625	4	15	60	6%	371	2	17.54	163.66
759375	5	15	75	6%	6952	3	759.04	2454.88

This section presented the assessment of BDD as a representation of formal contexts. It was observed that the BDD has a satisfactory performance only on context with fewer attributes and a large amount of objects, i.e., when the number of objects covers much of the maximum number of objects possible ($2^{|M|}$). Unfortunately, to achieve this exorbitant amount of objects, the number of attributes must be very small. Moreover, as mentioned, the performance of BDD deteriorates as the number of attributes increases. As the level of depth in the BDD graph increases, less simplification are found to reduce its size. The construction of the BDD is also affected when the time for its assembly grows exponentially when more attributes are expressed in the context. In addition, better results can be obtained in contexts with densities closer to the minimum and maximum values than in intermediate values.

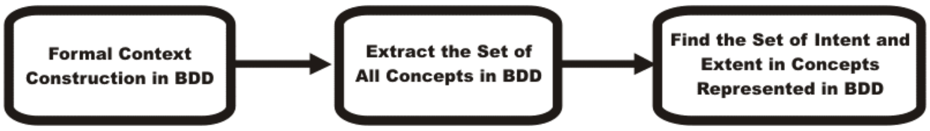
It is important to emphasize that the FCA derivation operator, necessary to obtain the formal concepts, is applied on the context expressed in BDD. Therefore, the more satisfactory is the performance of BDD, smaller will be the computational time required to operate it. For this reason, the concepts extraction should take advantage of this situation.

4 Formal Concepts Extraction using BDD

4.1 Concept Extraction Algorithm Implementation Using BDD

In order to use a BDD representation of formal context, 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]. Note that the purpose of this paper is to evaluate the feasibility of BDD and not its most efficient implementation over several others algorithms.

The implementation of the Attribute Intersection algorithm in BDD was divided in three primary stages (Fig. 6). In the first stage, the construction of the formal context in BDD is made (Algorithm 1). The second stage is responsible to extract the set of all concepts from the BDD context. In the final stage, it is necessary to identify the attributes and objects from the concepts represented in BDD.



Formal Context Construction in BDD
Extract the Set of All Concepts in BDD
Find the Set of Intent and Extent in Concepts Represented in BDD

Fig. 6. Steps to implement the Attribute Intersection algorithm in BDD.

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 that has an attribute. The intersection between two concepts is also implemented through an implicit BDD operation. In this case, the conjunction operator is represented in the algorithm as “&” but implemented as *bdd_and*. Moreover, the concepts list in this algorithm was implemented as a Hash to achieve a faster verification of concepts duplicity.

Algorithm 2. BDD construction based on the context.

```
in:  BDD context
out: List<BDD> concepts
concepts = new List<BDD>
concepts.addConcept(context)
for i=0; i<attributes; i++ do
  BDD tmp1 = context & bdd_ithvar(i)
  size = concepts.size()
  for j=0; j<size; j++ do
    BDD tmp2 = concepts.getConcept(j)
    BDD intersection = tmp1 & tmp2
    if !concepts.exist(intersection) then
      concepts.add(intersection)
    endif
  done
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 through the formal context, thereby maintaining the essence of the proposed Algorithm 2.

Finding the Set of Intent and Extent in Concepts Represented in BDD. This section shows how to obtain the extent and intent of these concepts represented in BDD. Algorithm 3 is used to check if all objects represented by the BDD share a common attribute. Algorithm 4 is used to verify whether or not an object is present in the BDD.

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

```
in:  BDD concept, attr
out: presence
BDD tmp = concept & bdd_ithvar(attr)
if tmp == concept then
  present = true
else
  present = false
endif
```

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 represented in BDD.

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

```
in:  BDD concept, objc
out: presence
BDD tmp = concept
for i=0; i<objc.attributes; i++ do
  if tmp == bddtrue then
    presence = true
    return
  else if tmp == bddfalse then
    presence = false
    return
  endif
  if bdd_varlevel(tmp) == i then
    if obj.hasAttribute(i) then
      tmp = bdd_high(tmp)
    else
      tmp = bdd_low(tmp)
    endif
  endif
done
presence =(tmp == bddtrue)
```

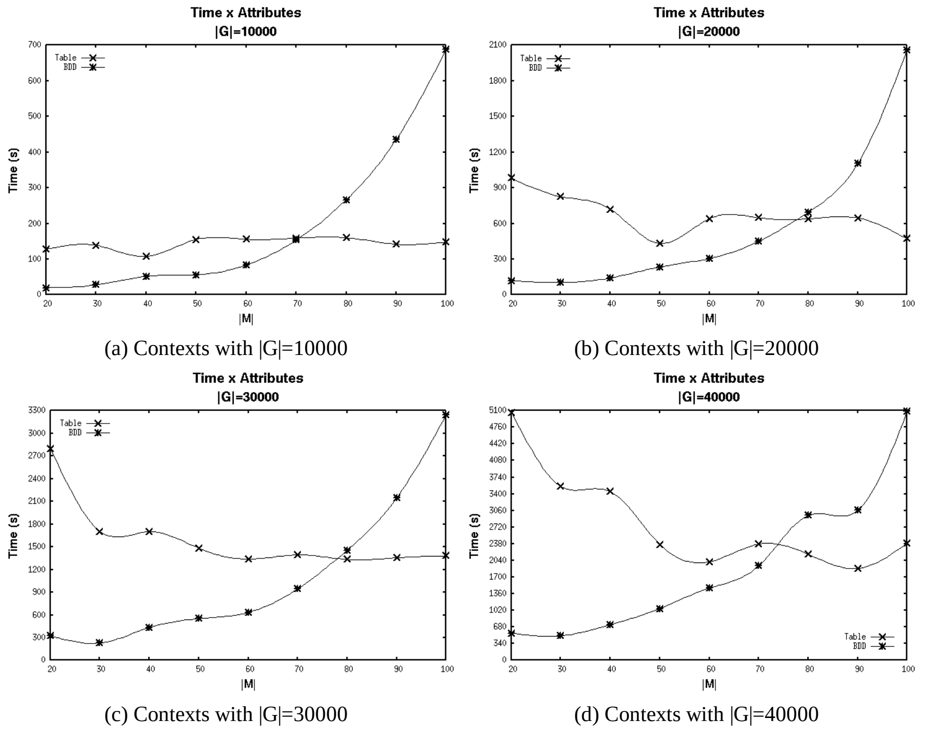
4.2. Feasibility Analysis of BDD to Extract Concepts

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 both situations: contexts represented by BDD and by a table. As previously mentioned, the algorithm selected was the Attribute Intersections. This algorithm choice was driven by its inherent characteristics that allow a more effectively concepts extraction from contexts where the number of objects is superior to the attributes.

To create a more reliable simulation environment, both versions of the algorithm were constructed sharing the same types of strategies. The BDD version was constructed according to the diagram in Figure 6, while the tabular version was constructed with several optimizations. Both of them uses a list that holds concepts intent as a hash-table and shares the same hash function. The BDD version has an intrinsically feature that, when there is an intersection between two other concepts, the result is already a concept; while in the tabular version it is necessary to further use derivation operators to acquire the concept. To overcome this problem, the concept is obtained only after the verification of if its intent is not present in the list yet. Thus, the tabular version of the algorithm avoids unnecessary derivation operations and maintains similarity to the BDD version. Another feature was the implementation in the conventional version of the algorithm: the derivation operator uses a data structure similar to a *BitSet*. The efficiency of the operators becomes superior by decreasing the amount of comparisons between two sub-sets of concept extents. In this sense, various enhancements aimed at a more rapid extraction of concepts in order to achieve a more effective comparison of the BDD viability.

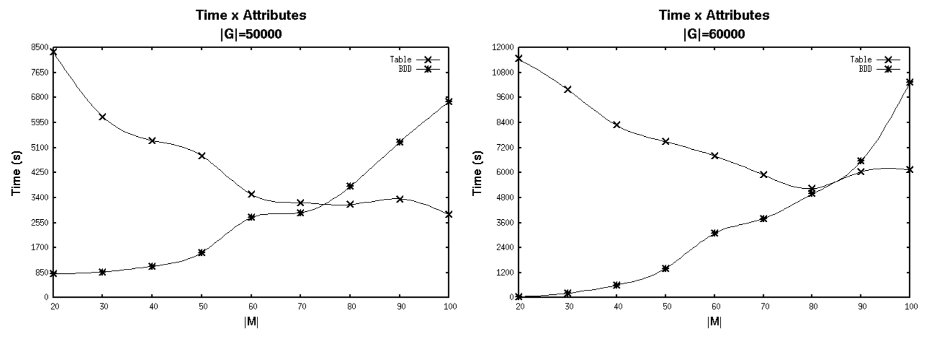
Another difference between the two versions evaluated is related to how each of them carries out their intersections. The BDD version performs the intersection between concepts represented in BDD through the implicit *bdd_and* operator, while the tabular version performs the intersection between the previously computed concepts extents. After that step, both algorithms must identify the concepts intent. The BDD version takes advantage of this situation because of its extremely efficient *bdd_ithvar* operator, but looses in performance in stage 3 of the diagram in Figure 6. The table version is not affected by this problem since it obtains the concept intent and extent through the derivation operators. Thus, several simulation scenarios are necessary to evaluate the algorithm behavior over different conditions.

Figure 7 shows the behavior of the Attribute Intersections algorithm for the BDD and tabular version for contexts with fewer attributes (20 to 100) and many objects (10,000 to 60,000). This algorithm has better performance for contexts $|G|>|M|$. To ensure that the BDD graph 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 consume less time to execute. All simulations were realized on a Pentium Dual Core 2.66Ghz with 2Gb of RAM running Linux Slackware Linux 12.0. The implementations for both versions of the Attribute Intersections algorithm were implemented in C++.



(a) Contexts with |G|=10000 (b) Contexts with |G|=20000

(c) Contexts with |G|=30000 (d) Contexts with |G|=40000



(e) Contexts with |G|=50000 (f) Contexts with |G|=60000

Fig. 7. Evaluation of Attribute Intersections implemented as a table and BDD.

As it can be seen in Figure 7, the implementation of the algorithm in BDD had an exponential performance in all the simulations, while the tabular version has presented an irregular decreasing behavior. For the table version, the density can explain the decreasing behavior, since lower attributes value had higher density for these considered simulations. In addition to that, how the incidences are spread into the context can explain its irregular behavior. Different contexts with the same density may have different execution performance. On the other hand, the BDD version presented a stable exponential behavior. Increasing the quantity of attributes in the context, more nodes will be required to construct the BDD. Therefore, as more nodes are used by the BDD, less efficient will be the operations in this representation. Thus, explaining this uniform behavior. Also, as can be seen by simulations of 20 and 30 attributes, while the tabular version had worst time performance, this BDD maintained a very low execution time, despite of the higher density. So, the BDD size is extremely relevant in the computation of all concepts.

Through simulation, it is demonstrated that BDD has a better overall performance than the table version for a number of attributes lower than or equal to 70. Above this threshold, the BDD graph becomes complex and begins to turn into an unattractive solution. Also, as the amount of objects increases, greatest has become the difference between the execution times of both implementations, considering attributes up to 70. 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. This time, the number of objects chosen was based on the ICFCA'06 challenge. A many-valued context was simulated with 7 attributes, a fixed number of 10 attribute-values per attribute and 120,000 objects. The context had a density of 10% and generated 1,172,960 concepts. Table 6 presents the spent time consumed by both algorithm implementations, in BDD and in table.

Table 6. Execution time for many-valued context with 70 attributes and 120000 objects.

	Construction of the Context (s)	Concepts Extraction (s)	Intent and Extent Identification (s)	Total (s)	Total
Table	-	-	-	251283	2d 21:48:03
BDD	128	18345	33289	51762	0d 14:22:42

As it can be seen in Table 6, the BDD version obtained the set of all concepts in less than 15 hours, while the tabular version demanded almost three days for its complete execution. Applicability to process larger contexts could be achieved with the use of a distributed version of an algorithm implemented in BDD. If we consider that a context with 70,000 attributes and 120,000 objects can be divided into sub-contexts of 70 attributes and 120,000 objects, still maintaining a low density, then it would be necessary, in general, 15 thousands of execution hours. Considering that all sub-contexts were executed in execution times around 15 hours. If a cluster of 50 computers were used, then it would be required around 300 execution hours, about 15 days. It will be still necessary to join the sub-concepts to form the concepts final set, but the BDD opens a possibility to process this large contexts.

Note that the required time to identify the set of extents from the concepts represented in BDD was very significant, as seen by Table 6. 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 identifying the objects, inverted form.

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.

In this work, the representation of the formal contexts in BDD were evaluated in two distinct aspects, as the memory consumption in relation to a bit table and as the computational time spent in its construction. It was later assessed the performance of the algorithm Attribute Intersection adapted to be used with BDD compared to the conventional implementation as a table. It has been verified that the BDD can be applied to the FCA algorithms to improve the execution time required to complete the extraction of all concepts. Although this representation allows the manipulation of contexts with a large number of objects, it is restricted to contexts with few attributes (up to 70 attributes, as experimentally verified). This is due to the fact that BDD tends to improve their representation with a larger number of objects, allowing further simplifications on its graph and thus making the operations on it more efficient. It has been also realized that the lower the number of attributes in the context the higher will be the BDD performance when compared to the conventional implementation of the algorithm, as verified in Figure 7. Thus, if the context meets this feature, a significant efficiency can be achieved with the application of this new alternative. This can also be verified for many-valued contexts in Table 6, in which the difference between both execution times was approximately of 2 days of uninterrupted processing.

The context density is an aspect that is intimately related to the number of concepts obtained. The concepts extraction using a BDD representation is still conditioned to this characteristic. Therefore, all simulations were limited to low densities.

Several future works may be pointed out: Evaluating new libraries for BDDs construction and manipulation, like CUDD [11]; measuring the behavior over different BDD technologies, like ZBDDs; evaluate different orders for attributes (statically or dynamically chosen) to construct the BDD; and adapting others FCA algorithms that could be used with BDD. ZBDD have already proven to be satisfactory for spare contexts [5], but in some type of contexts, in our preliminary results, the standard BDD was able to beat the ZBDD performance. Further analysis is therefore required. Also, in order to adapt others FCA algorithms a study must be conducted to verify whenever the BDD can be applied. In other words, which algorithm operations can be similarly replaced by a BDD operation in order to increase the algorithms capabilities. For example, in this work, the intersection of concepts in the Attribute Intersection was implemented by replacing this function with a correlated BDD conjunction operator that enabled performance improvements.

Although the results presented in this paper have been shown to be satisfactory for many objects (120,000) and a few attributes (in the order of 70), it is possible to use the BDD approach in conjunction with distributed FCA algorithms. Thus increasing processing power of contexts with larger number of attributes while still maintaining its inherent capability of processing huge amounts of objects.

References

1. Li, Y., Liu, Z.T., Shen, X.J., Wu, Q., Qiang, Y.: Theoretical research on the distributed construction of concept lattices. International Conference on Machine Learning and Cybernetics, 2003 1 (2003) 474-479 Vol.1
2. Liu, Z., Li, L., Zhang, Q.: Research on a union algorithm of multiple concept lattices. In: RSFDGrC, Springer (2003) 533-540
3. Lévy, G., Baklouti, F.: A distributed version of the ganter algorithm for general galois lattices. In: CLA 2005. (2005) 207-221
4. Bryant, R.: Graph-based algorithms for boolean function manipulation. IEEE Transactions on Computers C-35(8) (1986) 677-691
5. Yevtushenko, S.: Computing and Visualizing Concept Lattices. PhD thesis, TU Darmstadt, Fachbereich Informatik (2004)
6. Minato, S.: Zero-suppressed BDDs for set manipulation in combinatorial problems. In: DAC '93: Proceedings of the 30th International Conference on Design Automation, New York, NY, USA, ACM (1993) 272-277
7. Grätzer, G.: General Lattice Theory. Birkhäuser, Basel (1978)
8. Butler, K.M., Ross, D.E., Kapur, R., Mercer, M.R.: Heuristics to compute variable orderings for efficient manipulation of ordered binary decision diagrams. In: DAC'91: Proceedings of the 28th Conference on ACM/IEEE Design Automation, New York, NY, USA, ACM (1991) 417-420
9. 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.
10. Carpineto, C., Romano, G.: Concept Data Analysis: Theory and Applications. John Wiley & Sons, Indianapolis, IN, USA (2004)
11. Somenzi, F.: CUDD: CU decision diagram package release (1998)