SCGaz – A Synthetic Formal Context Generator with Density Control for Test and Evaluation of FCA Algorithms

Andrei Rimsa[1], Luis E. Zárate[2], Mark A. J. Song[2]

[1]Departament of Computer Science
Federal Univeristy of Minas Gerais (UFMG) – Belo Horizonte, MG – Brazil

[2]Computer Science Department, Applied Computational Intelligence Laboratory
Pontifical Catholic University of Minas Gerais – Belo Horizonte, MG – Brazil

rimsa@dcc.ufmg.br, {zarate,song}@pucminas.br

Abstract. *An efficient way to evaluate FCA algorithms is through a comparative analysis of their performance in typical contexts. Comparisons are normally conducted using randomly generated contexts that may contain duplicated attributes and objects and other types of redundancies. Failing to acknowledge the presence of these redundancies in formal contexts could lead to erroneous comparison analysis. This paper proposes a tool named SCGaz (Synthetic Context Generator) that randomly fills synthetic formal contexts ensuring the absence of some type of redundancies. At the same time, the tool is able to keep track of the contexts density, allowing users to select any density in the bounds of the minimum and maximum permitted for a type of context. Thus, this approach allows more controllable and reliable simulation environment. In this work, an analysis of the time spent to generate different types of formal contexts, including large ones, is presented. As a case study, a performance comparison between Object Intersection algorithm and its dual version, Attribute Intersections, with contexts generated by SCGaz is discussed. Contexts produced by SCGaz in conjunction with real world dataset allow a more in-depth comparative analysis of FCA algorithms performance.*

1. Introduction

Formal Concepts Analysis (FCA) [1,4,5,9] is an applied mathematical field whose main objective is to represent knowledge through specific structures called line diagrams or lattice. In recent years, the FCA has been used as a powerful technique to represent and extract knowledge from datasets expressed as cross tables, namely formal contexts. A *Formal Context (G, M, I)* is a triple, where *G* is the set of objects denoted extension, *M* is a set of attributes denoted intention, and *I* is a set of incidences $(I : G \rightarrow M)$.

Since each element of the extent has elements of the intent, and element of the intent is also an attribute of the elements in the extent, it is possible to create a partial order on the set of concepts: $(A_i, B_i) \leq (A_{i+1}, B_{i+1})$, $A_i \subseteq A_{i+1}$, $B_i \subseteq B_{i+1}$. The concepts partially ordered are called concept lattice.

With the increase interest of FCA as a technique to represent and extract knowledge, many algorithms were proposed in the literature. For example, in [11] a comparison between several of them is presented. Sometimes algorithms were created

envisioning computational performance improvements to extract concepts or to build the lattice, other times to implement new extraction dependency operators. However, it is a well known fact that FCA algorithms have different behaviors on different datasets (contexts). Due to these difficulties to achieve a more controlled experimental comparison environments, Kuznetsov (2004) proposed two alternatives for dataset usage as test beds for FCA algorithms evaluation: (1) "real databases" consisting of open databases recognized by the FCA community in which several patterns can be specified; and (2) random generated contexts, where the randomness strategies must be known. The present work is related to that latter.

Usually, the generation of random formal contexts seeks to encounter a desired density or the probability that an object has a given attribute. Still, another approach used in [7] defined a fixed amount of attributes for each object randomly generated, i.e., all the objects have the same number of attributes. However, neither of them have a control over redundant objects (two or more similar objects) or attributes that could be target for reduction (attributes with incidences on all objects). Controlling theses aspects is an important step to obtain a more reliable comparative analysis of FCA performance algorithms. Before the application of any algorithm to extract concepts, a context could be pre-processed in order to remove attributes and objects that would not contribute to the final concept lattice or the final set of all concepts. The tool ConExp [3] is an example that allows the reduction of attributes and objects from existing contexts. This procedure could improve considerably the concept extraction process, mainly for the case of large contexts [2]. However, this pre-processing alters the previously context density, which cannot be know after the process is completed. Another way to build a context without redundant objects is to generate all possible objects $[1..2^{|M|}]$ and select some objects from that set. This approach ensures no repeated objects, but it loses control over the density, which could be reached earlier (or later) for the amount of objects required. By neglecting the existence of redundant attributes and objects, FCA algorithms may hide or mask its true efficiency. Also, control over the density leads to more controlled environment to evaluate FCA algorithms behavior.

This paper proposes a tool SCGaz [8] to create randomly filled synthetic formal contexts, bounded by a minimum and maximum density for each type of context selected by a user. The control over contexts have the objective of creating partially irreducible contexts, that is, contexts that have no redundant objects nor attributes with incidences on all objects or objects/attributes without incidences. Although the presence of objects and attributes resulting from the intersection between other objects and attributes is allowed by SCGaz, in real world applications, they tend to be specific to a certain domain and should not be removed from the context.

For type of contexts were considered in this work: $|G|=|M|$, $|G|<|M|$, $|G|>|M|$ and many-valued contexts. For each context, the computational time is calculated to determine the feasibility of such approach based on the context size (number of attributes and objects) and its density. The aim is to improve the reliability of experimental simulations comparing algorithms using partially irreducible contexts.

This article is divided in 6 sections. In section 2, a review of Formal Contexts is presented. In section 3, the algorithms used for the generation of synthetic formal

contexts are proposed. In section 4, a case study is provided. Finally, in the last sections the conclusions and future works are discussed.

2. Formal Context

Formal contexts have the notation $K:=(G, M, I)$, where G is a set of objects, M is a set of attributes 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, this 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:

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

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

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

$$A \subseteq G = \emptyset \Rightarrow A' := M \; ; \; B \subseteq M = \emptyset \Rightarrow B' := G \quad \textbf{(3)}$$

Even though this definition of formal contexts is valid for many situations, mainly to represent objects that have the presence or absence of some properties (attributes), it is not a good representation for the major part of situations where objects have attributes that can take on several values. For this case, attributes are called many-valued attributes. So, contexts where the set M of attributes is composed by many-valued attributes are called many-valued contexts. The notation of the formal context is given by the quadruple (G, M, V, I), where G is a set of objects; M is a set of many-valued attributes; V is the set of attribute-values; and I is a ternary relation between G, M and V ($I \subseteq G \times M \times V$). In general, many-valued contexts can be transformed into single-valued contexts in order to obtain the formal concepts. A simple way to do such transformation is to replace each many-valued attribute by the corresponding attribute-value pair. Note that the number of attribute-value pairs for each many-valued attribute is given by the cardinality of the set V (i.e. $|V|$). It is important to remember that the attributes can have continuous values. So, for this case, it may be interesting to choose intervals of values for the attributes.

3. Synthetic Formal Context Generation

3.1. Typical Formal Context Analysis

In this work, formal contexts were selected in order to cover the largest range of real application possible. The contexts considered are: $|G|=|M|$, $|G|<|M|$, $|G|>|M|$ and many-valued contexts. The minimum and maximum densities related to each type are analyzed for a defined number of objects and attributes. Like that, it is ensured that a user will not choose a density outside this range. A set of rules based on FCA fundaments could be established in order to generate the synthetic formal contexts.

Theorem 1: Given a formal context $K: = (G, M, I)$ and its formal concept $\underline{\beta}(G, M, I)$ if $\underline{\beta}(G, M, I) \approx \underline{\beta}(G, M - \{m\}, I \cap (G \times (M - \{m\})))$, then m is a attribute reducible.

Corollary 1: Considering $m \in M$, $I_m \subset I$, and $I_m = G \times \{m\}$, if $I_m = \{\ \}$, then m is attribute reducible.

Corollary 2: Considering $m \in M$, $I_m \subset I$, and $I_m = G \times \{m\}$, if $|I_m| = |G|$, then m is attribute reducible.

Theorem 2: Given a formal context $K: = (G, M, I)$ and its formal concept $\underline{\beta}(G, M, I)$, if $\underline{\beta}(G, M, I) \approx \underline{\beta}(G - \{g\}, M, I \cap ((G - \{g\}) \times M))$, then g is object reducible.

Corollary 3: Considering $g \in G$, $I_g \subset I$ and $I_g = \{g\}$ x M, if $I_g = \{\ \}$, then g is object reducible.

Corollary 4: Considering $g \in G$, $I_g \subset I$ and $I_g = \{g\} \times M$, if $|I_g| = |M|$, then g is object reducible.

Definition 1: If $\exists\, g, h \in G$ such that $g' = h'$ then $g = h$. Accordingly, for $m, n \in M$, $m'=n'$ implies $m = n$. The context is then called clarified. As a result g or h is reducible.

The **Theorems 1** and **2** define respectively what a reducible attribute and object is, that is, attributes and objects that can be removed from the context with no impact on the lattice structure. It is important to note that an attribute can be removed from the formal context if it is not shared by any objects (**Corollary 1**) or, on opposite, when all objects share it (**Corollary 2**). The same applies to any object that does not have any attributes (**Corollary 3**) or, on opposite, that has all the attributes (**Corollary 4**). When two objects share the same attributes, FCA considers them conceptually identical (redundant). Hence, these objects can be reduced to a single one (**Definition 1**). The SCGaz considers all these restriction when generating formal contexts. The main formal contexts considered in this work are now presented with their respectively minimum (D_{min}) and maximum (D_{max}) density values. Where $g=|G|$, $m=|M|$ and $v=|V|$.

a) g = m

Table 1. Minimum Density

	a_1	a_2	a_3	a_4	a_5
o_1	X				
o_2		X			
o_3			X		
o_4				X	
o_5					X

Table 2. Maximum Density

	a_1	a_2	a_3	a_4	a_5
o_1		X	X	X	X
o_2	X		X	X	X
o_3	X	X		X	X
o_4	X	X	X		X
o_5	X	X	X	X	

$$D_{\min} = \frac{1}{g} \quad \textbf{(4)}$$

$$D_{\max} = \frac{g-1}{g} \quad \textbf{(5)}$$

b) g < m

Table 3. Minimum Density

	a_1	a_2	a_3	a_4	a_5
o_1	X			X	
o_2		X			X
o_3			X		

Table 4. Maximum Density

	a_1	a_2	a_3	a_4	a_5
o_1		X	X		X
o_2	X		X	X	
o_3	X	X		X	X

$$D_{\min} = \frac{1}{g} \quad \textbf{(6)}$$

$$D_{\max} = \frac{g-1}{g} \quad \textbf{(7)}$$

c) g > m

Table 5. Minimum Density

	a_1	a_2	a_3	a_4
o_1	X			
o_2		X		
o_3			X	
o_4				X
o_5	X	X		
o_6		X	X	
o_7			X	X

Table 6. Maximum Density

	a_1	a_2	a_3	a_4
o_1		X	X	X
o_2	X		X	X
o_3	X	X		X
o_4	X	X	X	
o_5			X	X
o_6	X			X
o_7	X	X		

$$D_{\min} = \frac{\sum_{i=1}^{h}(C_i^m \times i) + R \times (h+1)}{g \times m} \Bigg| R = g - \sum_{i=1}^{h} C_i^m \quad and \quad \max(h) \rightarrow R > 0 \quad \textbf{(8)}$$

$$D_{\max} = \frac{\sum_{i=1}^{h}[C_{(m-i)}^m \times (m-i)] + R \times (m-h-1)}{g \times m} \Bigg| R = g - \sum_{i=1}^{h} C_{(m-i)}^m \quad and \quad \max(h) \rightarrow R > 0 \quad \textbf{(9)}$$

d) Many-Valued

$$D_{\min} = D_{\max} = \frac{m}{v} \quad \textbf{(10)}$$

Table 7. Minimum and Maximum Density

	a_1		a_2			a_3			
	i_1	i_2	i_3	i_4	i_5	i_6	i_7	i_8	i_9
o_1	X		X			X			
o_2		X		X			X		
o_3	X				X			X	
o_4		X	X						X
o_5	X			X		X			

All rules of section 3.1 are ensured for attributes ($a_1..a_m$) and objects ($o_1..o_n$) in many-valued contexts, but are not extended for many-valued attributes ($i_1..i_V$).

Table 8 shows the relation between the minimum and maximum number of objects for all four considered contexts. Ensuring these minimum and maximum conditions is an important step to obtain a controlled context. It is important to emphasize that in the SCGaz tool each context is automatically detected based on input data (number of objects and attributes). This is due to the fact that each context requires a different procedure for its correct filling.

Table 8. Minimum and maximum of objects for each considered context.

Context Type	Objects Minimum	Objects Maximum
g = m	m	m
g < m	2	m – 1
g > m	m + 1	$2^m - 2$
Many-Valued	1	$\prod_{j=1}^{m} \lvert a_j \rvert$

3.2. Synthetic Formal Context Generation with Density Control

In this section, the algorithms related to the initialization and filling process to build the formal contexts are presented.

Initialization of Formal Contexts. In order to generate the synthetic context, it is primarily important to determine its type (section 3.1) to calculate its minimum density. For each context, a distinct procedure must be used to ensure the absence of reducible attributes and objects (Theorems 1 and 2). Figure 1 illustrates the initialization process.

Fig. 1. Diagram for Formal Context Initialization.

Several modules have been developed for the SCGaz tool. The algorithm's kernel is described as follow. Algorithm 1 shows the pseudo-code for the initialization process of synthetic contexts. Initially, the algorithm identifies each type of context that will be used in the initial filling process based on the number of objects and attributes specified by a user. The objects are created according to a specific subroutine for each context described in section 3.1. Objects are generated by the *addObject* operation, in which it tries to add the object into a randomly selected line on the table. Thus, if an object is successfully added to the context, then another one will be generated for further filling. Otherwise, if a newly generated object is already present in the context, then another object, semantically identical (the same number of attributes), must be generated again. In contexts with more objects than attributes, identical objects can repeatedly be generated. For this reason, another distinct object must be generated in order to continue the initial filling of the context. This process is repeated until all objects are generated according to the minimum density value defined for the formal context. The algorithms and subroutines are available from [8]. It is important to notice that the subroutines were created to avoid the creation of inconsistent objects, according to the rules earlier defined in section 3.1.

Algorithm 1. Synthetic Context Initialization

```
 1: in attributes, objects
 2: ctx = new SyntheticContext
 3: type = selectContextType(attributes, objects)
 4: while type->hasMoreObjects() do
 5:   obj = type->getObject();
 6:   if ctx->addObject(obj) then
 7:     type->nextObject()
 8:   endif
 9: done
10: return ctx
```

Filling Formal Contexts. After the minimum density value for the context is reached, the next step is to continue its filling until it reaches a user selected density, the maximum density value or until there are no more cells in the context to be used. Note that this process doesn't need to meet Corollaries 1 and 3 since it has already been met during the context initialization. The rest of the filling process is given by the pseudo-algorithms below represented.

Note that this process doesn't need to meet Corollaries 1 and 3 since it has already been met during the context initialization. The rest of the filling process is given by the pseudo-algorithms below represented.

Algorithm 2. Random filling of the context controlling the density

```
 1: in D_desejado
 2: in ctx
 3: assert (ctx->D_min) ≤ D_desired ≤ (ctx->D_max)
 4: max = false
 5: while ctx->density < D_desired and !max do
 6:   if !bruteFilling(ctx, ATTEMPTS_MAX) then
 7:     if !smartFilling(ctx) then
 8:       max = true
 9:     endif
10:   endif
11: done
```

Algorithm 2 shows the main idea to continue the filling of the context at a higher level of abstraction, where two auxiliary algorithms were used. Algorithm 2.2 (*Smart filling*) presents a smarter way to randomly filling the contexts cells. It keeps tracks on those cells that were already selected for marking and could not be previously used. By doing so, the algorithm avoids selecting unusable cells and can randomly select the remaining ones that still could be used. However, this last algorithm has a high computational cost. To improve the filling process, another algorithm has been proposed, Algorithm 2.1 (*Brute Force Filling*). It uses a brute force approach to fill each cell in a faster way. Each loop of Algorithm 2 is responsible for the creation of one incidence on the context. The process continues until a chosen density is achieved or until the contexts has no more usable cells. Details about the auxiliary algorithms are presented below.

Algorithm 2.1. Brute force filling of the context.

```
 1: function bruteFilling
 2:   in ctx
 3:   in attempts
 4:   for x = 0 ; x < attempts ; x = x + 1 do
 5:     attr = random % ctx->attributes
 6:     objc = random % ctx->objects
 7:     if ctx->setPosition(attr, objc) then
 8:       return true
 9:     endif
10:   done
11:   return false
12: endfunction
```

First, the "*Brute Force Filling*" algorithm is responsible for the best performance in the context filling. Positions (i, j) in the table are randomly sorted for its possible use. Each position must be checked for availability and if it is in accordance with the definitions and corollaries described in section 3.1. Cells are continually sorted until an able cell position is found or until it reaches a certain number of iteration (in this tool, 10,000 iterations). If the maximum iterations value is reached, then the algorithm

"*Smart Filling*" is activated, according to Algorithm 2. Although Algorithm 2.1 has lower overhead in the filling process, it can hardly converge to the maximum density. This is due to the low likelihood of sorting a free usable cell – it decreases as the density increases. This algorithm has another disadvantage. It cannot determine when there are no more usable cells on the context.

Algorithm 2.2: Smart Filling of the context.

```
 1: function smartFilling
 2:   in ctx
 3:   list<Object> objc = ctx->allObjects()
 4:   while objc->size > 0 do
 5:     objc_pos = random % objcs->size
 6:     objc = objs->removeElement(objc_pos)
 7:     list<Attribute> attrs = obj->unsettedAttributes()
 8:     while atrrs->size > 0 do
 9:       attr_pos = random % atrs->tamanho
10:       attr = attrs->removeElemento(attr_pos)
11:       if ctx->setPosition(attr->pos, objc->pos) then
12:         return true
13:       endif
14:     done
15:   done
16:   return false
17: endfunction
```

Algorithm 2.2, as previously mentioned, allows a smarter filling on the context, while still maintaining its randomness. A list containing all objects in the context is created where objects are being sorted and removed from it. For each object removed from the list, another list is created that holds unmarked attributes for that object. Again, attributes are being sorted and removed from that list forming the context cells position, *par(attribute, object)*. The assembled *par* is submitted for marking by the *setPosition* function.

Due to the restrictions imposed during the filling process, one selected density value may never be reached. There are situations where there are no possible markings to increase the contexts density, requiring a forced algorithm interrupt. These situations happen only for density values near the maximum. For that reason, the achieved density is then considered the maximum for that context.

3.3. Synthetic Context Generation Analysis

In this section, the computational time required to generate the context for the four scenarios proposed before is presented. Three density values, for each combination of attributes and object, were considered: D_{min}, $D_{50\%}$ and D_{max}. Besides for many-valued contexts, that has a single fixed density. The simulations were made on a Pentium IV 3.06 GHz with HT 2Gb of memory. The simulation results are presented in Tables 9, 10, 11 and 12.

Table 9. Simulation for Contexts $|G|=|M|$.

Attributes	Objects	Time (s) for D_{min}	Time (s) for $D_{50\%}$	Time (s) for D_{max}
100	100	0.00	0.03	0.06
500	500	0.01	2.67	5.53
1000	1000	0.05	20.97	42.79
2000	2000	0.23	169.18	342.29
3000	3000	0.87	635.24	1297.00
4000	4000	2.34	1449.41	3065.49
5000	5000	4.54	4856.03	9401.58

Table 10. Simulation for Contexts $|G|<|M|$.

Attributes	Objects	Time (s) for D_{min}	Time (s) for $D_{50\%}$	Time (s) for D_{max}
100	50	0.00	0.01	0.02
500	250	0.01	0.67	1.43
1000	500	0.01	5.28	10.97
2000	1000	0.04	42.41	86.42
3000	1500	0.10	145.42	291.14
4000	2000	0.20	343.58	700.92
5000	2500	0.38	698.32	1453.11

Table 11. Simulation for Contexts $|G|>|M|$.

Attributes	Objects	Time (s) for D_{min}	Time (s) for $D_{50\%}$	Time (s) for D_{max}
100	200	0.01	0.09	0.19
500	1000	0.08	10.59	21.58
1000	2000	0.43	84.41	170.23
2000	4000	2.61	988.73	2112.53
3000	6000	12.03	6001.92	12780.21
4000	8000	28.95	17732.17	35577.00
5000	10000	47.10	37028.39	72686.15

Table 12. Simulation for Many-Valued Contexts

Attributes	Attribute-Values	Objects	Density (%)	Time (s)
10	10	10000	10.00	2.14
10	20	10000	5.00	2.23
10	30	10000	3.33	2.43
10	40	10000	2.50	2.51
10	50	10000	2.00	2.93

The algorithms have had an exponential time for each type of context considered. The density values have a direct impact on the computational time. The higher the density value, the higher will be the necessary time to create the context. Contexts with lower density value, such as the many-valued one, have a better overall performance.

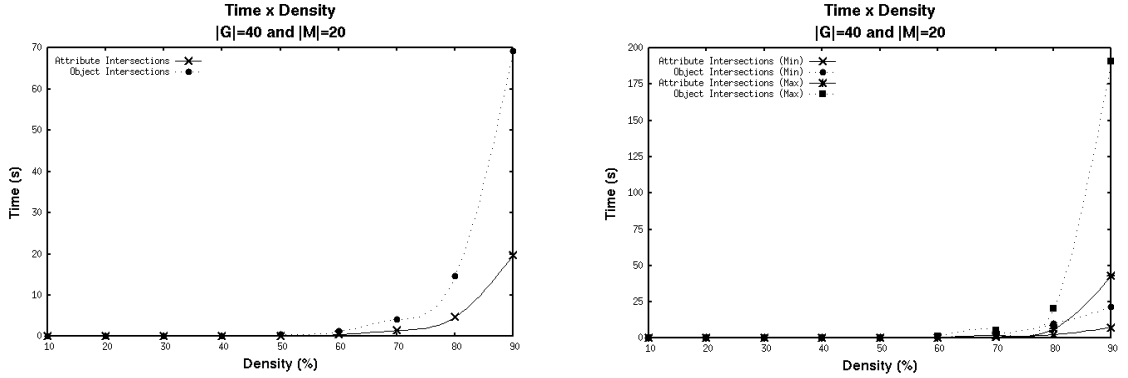
It is important to notice that contexts with high number of objects tend to have the worst performance. Because of **Definition 1**, each object must be compared with the others on the context to ensure no duplicity. Thus, as the number of objects grows, the comparisons between those objects grow exponentially.

4. Case Study

To demonstrate the feasibility of the SCGaz as a tool for synthetic contexts generation, two different algorithms were evaluated: Object Intersections, discussed in [1] and its dual version, Attribute Intersections. They were both evaluated over three different types of contexts and with densities ranging from lower to higher values. It is noteworthy that both use the incremental intersections strategy to compute the formal concepts.

Those two algorithms have a disadvantage because they fail to prevent the calculations of concepts already computed previously. A newly obtained concept must be verified in relation to duplication before it can be added to the concepts list. Therefore, to improve the search for concepts on that list, a hash table has been used, implemented with collisions resolved by linked lists. In such cases, the complexity related to the Attribute Intersections and Object Intersections algorithms is respectively: $O(|M||C||K|)$ and $O(|G||C||K|)$. Where C is the list containing all concepts and K is the sub-lists of $|C|$ that are dependent on the hash function used. Thus, $1 <= |K| <= |C|$, where K should be ideally one (1).

With the purpose of simplicity, only 3 types of contexts have been considered in the evaluation: $|G|>|M|$, $|G|<|M|$ and $|G|=|M|$. The algorithms have been implemented in C++ and both share the same software structure, including the same derivation operator and hash function. In the experiments, densities have been selected ranging from 10% to 90%. For each density value, one hundred (100) random contexts have been generated. The average, minimum and maximum execution times related to each algorithm are showed in Figures 2, 3 and 4. The simulations have been performed on a Pentium IV 3.06 GHz with HT 2Gb of memory.

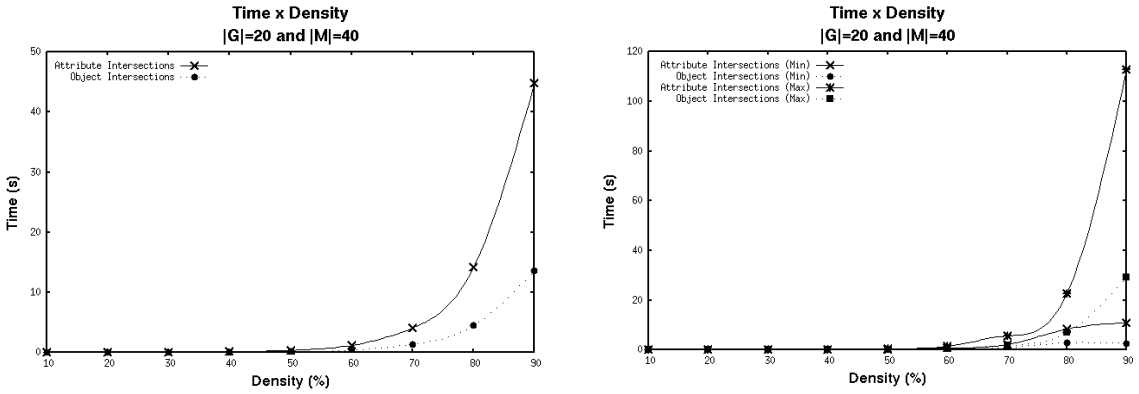


(a) Average times. (b) Minimum and maximum times.

Fig 2. Attribute and Object Intersection for contexts |G|>|M|, where |G|=40 and |M|=20.

In contexts where $|G|>|M|$, the Attribute Intersections algorithm has had a better performance, Figure 2(a). For densities below 60%, the computations of all concepts have spent approximately 1 second for both algorithms. However, in higher densities, the number of concepts generated has increased and therefore the number of intersections needed has increased, resulting in a higher computational time cost. In these situations, a great discrepancy between the two algorithms can be seen.

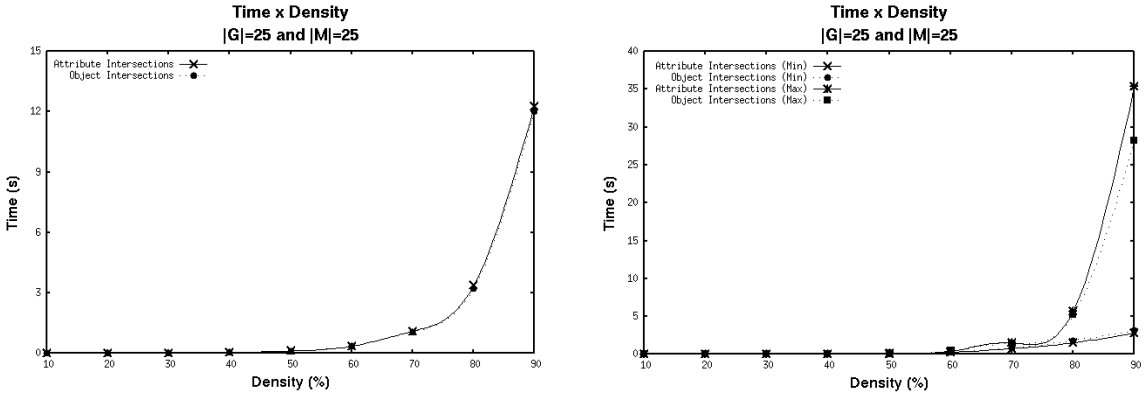
According to the algorithms complexity, it is expected that the Object Intersection algorithm presents a worse performance for context with more objects than attributes, Figure 2(a). However, it is observed in Figure 2(b) that the algorithm in some occasions has had better results compared with its dual version. But as noted, this only occurs for fewer of the contexts evaluated.



(a) Average times. (b) Minimum and maximum times.

Fig. 3. Attribute and Object Intersection for contexts |G|<|M|, where |G|=20 and |M|=40.

Figure 3(a) shows the expected behavior for both of the algorithms. In this case, the Object Intersections algorithm has been more efficient than its dual version for contexts with |G|<|M|. It can be observed experimentally that the algorithms behave correctly according to their order of complexity. Again, there were contexts where the Attribute Intersections algorithm has a better performance, Figure 3(b), over its dual version, but in general, the use of the Object Intersections algorithm is more adequate.



(a) Average times. (b) Minimum and maximum times.

Fig. 4. Attribute and Object Intersection for contexts |G|=|M|, where |G|=25 and |M|=25.

The last experiment has considered contexts in which $|G|=|M|$. This time, both algorithms have the same order of complexity. Thus, their average execution time is extremely similar, as verified in Figure 4a. For the minimum and maximum times, there is a certain different between them, as seen in Figure 4b. Considering the maximum time limit, the Object Intersections algorithm has had a better performance. Considering the minimum time limit, the Attribute Intersections algorithm has been slightly more efficient. Choosing the best algorithm to be applied in these situations is a difficult task. The implementation of both becomes very sensitive to the context incidents, rather than the number of attributes and objects.

Through simulations, it is possible to conclude that the Attribute Intersections algorithm should be used when context has $|G|>|M|$. The Object Intersections algorithm should be used when $|G|<|M|$. When $|G|=|M|$, choosing the most efficient algorithm becomes a more difficult task, since both have practically the same behaviour. This choice, for this last type of context, should be driven based on the progression of the concepts added to the list. Unfortunately, to previously determine in which progression the concepts are added to the list depends on how the incidences are distributed in the context. Heuristics must be created, applied to contexts, which could help identifying in advance some algorithm execution tendency. Efforts in this sense must be taken to select which of those two algorithms should be applied to contexts $|G|=|M|$. The analysis regarding these algorithms was possible thanks to a contexts generation environment, which permits the evaluation of them in especial conditions such as the density.

5. Conclusion

This work presents a set of strategies used to create a simulation environment to generate partially irreducible contexts while controlling their density values. An analysis of computational time spent to create different contexts has been provided. It has been verified the possibility to create contexts in a reasonable computational time, even for large contexts. This work embraces Kuznetsov (2004) proposal to count on randomly generate formal contexts as testbeds to evaluate FCA algorithms. In this work, a new tool, SCGaz, is conceived to generate partially irreducible formal contexts.

Another way to create irreducible contexts consists in generating randomly uncontrolled formal contexts (considering or not a density value) to be latter applied to a context reduction algorithm. However, if applied, the density will be calculated a posteriori, losing control over the simulations environments. The reduction will also affect the amount of both attributes and objects, thus creating an unpredictable behavior of the resulting context. Therefore, the main advantage of the SCGaz is to create partially irreducible contexts with the amount of attributes, objects and density defined before its creation, i.e. *a priori*. Because of that, a more effective control over the simulations and analysis can be done.

Another important aspect of this work is not to consider fully irreducible contexts. Some of the objects and attributes that could be removed from a context may be truly meaningful to a specific application study. Once removed, those objects and attributes cannot be recovered from the built conceptual lattice.

As a case study, it has been used contexts generated by the SCGaz tool to evaluate two algorithms used to extract concepts: Object Intersections and Attribute Intersections. Through the use of this tool, it was possible to evaluate intervals of performance for both algorithms. As a result, it has been shown that the best algorithm to be applied must be selected based on the context features**,** such as objects, attributes and density.

6. Future Work

If new restrictions are imposed to the context, then these contexts could be considered irreducible. If two or more attributes have the same implication set (identical columns), then a new restriction could be specified. They could be reduced to a single attribute (**Definition 2**).

Definition 2: If $\exists\, m, n \in M$ such that $m' = n'$ then $m = n$. Accordingly, for any $g, h \in G$, $g' = h'$ implies in $g = h$. The context is therefore called **clarified**. Thus m or n can be reducible.

As mentioned before, the ConExp tool [3] uses another reducing strategy that still maintains the lattice structure. If an intersection between any objects result in another object present in the context, this one could be removed (**Corollary 6**). The same applies for attributes (**Corollary 5**).

Corollary 5: Considering $\forall\, m \in M$ and $\exists\, n,o \in M$, such that $n \cap o = m$ for $m \neq n \neq o$, then m attribute reducible.

Corollary 6: Considering $\forall\, g \in G$ and $\exists\, h,i \in G$, such that $h \cap i = g$ for $g \neq h \neq i$, then g is object reducible.

References

1. Carpineto, C.; Romano, G. Concept Data Analysis: Theory and Applications. Indianapolis, IN, USA: John Wiley & Sons, 2004. ISBN 0470850558.
2. Cole, R. J.; Eklund, P. W.; Groh, B. Dealing with Large Contexts in Formal Concept Analysis: A Case Study Using Medical Texts. In: International Symposium on Knowledge Retrieval, Use, and Storage for Efficiency, KRUSE-97. Berlin, Heidelberg: Springer Verlag, 1997. p. 151–164.
3. CONEXP, Yevtushenko, S., et. al: “Concept Explorer”, Open Source Java Software Avaliable at http://sourceforge.net/projects/conexp, Release 1.3, 2008.
4. Ganter, B.; Stumme, G.; Wille, R. Formal Concept Analysis, Foundations and Applications, v. 3626 de Lecture Notes in Computer Science, (Lecture Notes in Computer Science). Berlin, Heidelberg: Springer, 2005. ISBN 3-540-27891-5.
5. Ganter, B.; Wille, R. Formal Concept Analysis: Mathematical Foundations. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1997. ISBN 3540627715.

6. Kuznetsov, S.O. Algorithms for Computing Closed Sets: A Review, Workshop Evaluation des Algorithms de Generation de Concept et de Regles, Clermont-Ferrand at the ECG Conference, January 2004.
7. Kuznetsov, S.O.; Obiedkov S.A. Comparing Performance of Algorithms for Generating Concept Lattices, Journal of Experimental and Theoretical Artificial Intelligence, vol. 14 (2002), pp. 189-216.
8. SCGaz, Rimsa, A., at. Al: SCGaz tool (Synthetic Context Generator), Open Source Software Available at http://www.inf.pucminas.br/projetos/licap, Release 0.8, 2008.
9. Wille, R. Formal Concept Analysis as Mathematical Theory of Concepts and Concept Hierarchies. In: Formal Concept Analysis. Springer Verlag, 2005. p. 1–33. ISBN 978-3-540-27891-7.
10. Burmeister, P. Formal Concept Analysis with ConImp: Introduction to the Basic Features. Technical Report, TU-Darmstadt, Germany, 1996.
11. Tilley, Thomas (2004). Tool Support for FCA. In: Eklund (ed.), Concept Lattices: Second International Conference on Formal Concept Analysis, Springer Verlag, 2004. LNCS 2961, p. 104-111.