Distributed and General Galois Lattice For Large Data Bases

Fatma Baklouti and Gérard Lévy

CERIA Laboratory, Paris Dauphine University
Place du Maréchal de Lattre de Tassigny, 75775 Paris cedex 16, France
{fatma.baklouti, glevy}@dauphine.fr

Abstract. Large data processing is an essential problem in many data mining application. Our work explores the algorithms calculating a Generalized Galois Lattice (G^2L) over a large collection of data. A G^2L contains all the so-called closed sets of a collection of tuples (individuals characterized by a set of properties), G^2Ls generalize to multivalued data the GL already popular in the concept analysis using the binary values. They appear an attractive tool for data mining. The G^2L or GL calculus is CPU intensive. In practice, the current techniques limit the approach only to small sets of data only, e.g., a hundred tuples with a few dozens of properties each. Our research consists in building scalable distributed G^2L calculus algorithms. We think this research direction promising and probably the only way towards our goal. We first have implemented and analyzed some centralized algorithms. One termed ELL seems to be the most efficient. We have defined therefore a scalable distributed version of ELL termed SD-ELL. It recursively partitions the set of tuples for the G^2L computation over sufficiently many sites to let ELL execute fast enough at each site. The closed sets produced by each site enter a common scalable distributed data structure (SDDS). We then plan to test the resulting behavior and analyze the performance of the algorithm and of some promising variants.

1 Introduction

A Galois Lattice (GL) allows extracting concepts and rules from other concepts. Several algorithms determine the GL over a given context C, provided the size of C is small, [4] [7] [8]. These algorithms apply to binary data. Their generalization, G^2L was proposed in [5]. An algorithm for G^2L calculus termed ELL, and some alternative ones, were subsequently defined in [6]. These algorithms, as any previous for a GL computation, were designed for the single-site computations. It was known that they could run in reasonable time only for small sets, e.g., at most of a few hundreds of tuples.

Further analysis has shown that the most promising way towards larger sets of data, the only of interest to the data mining, was to find some scalable distributed variant of G^2L calculus. This goal becomes the subject of our work.

We have defined therefore a scalable distributed version of ELL termed SD-ELL. It recursively partitions the set of tuples for the G^2L computation over sufficiently many

Radim Bělohlávek, Václav Snášel (Eds.): CLA 2005, pp. 197–206, ISBN 80–248–0863–3.

sites to let ELL execute fast enough at each site. The closed sets produced by each site enter a common scalable distributed data structure (SDDS). We then plan to test the resulting behavior and analyze the performance of the algorithm.
The rest of the paper is organized as follows. In Section 2, we recall the definition of a G^2L. Section 2 recalls the main idea in ELL. Section 3 overviews the principles of our scalable distributed extension, exploring the recursive G^2L calculation in [6]. Section 4 concludes this paper by listing the future research direction.

2 General Galois Lattices

Concept lattice [9] and Closed itemset lattice are based on order theory and lattice theory [3]. They are used to represent the order relation on concepts or closed itemsets. Concept lattice describes the character of the set pair: intent and extent of concept. And the general of Galois Lattice formalism was addressed by [16], [17].
In this section, we define some notions generalizing usual ones: Data context, Closure operator, Closed itemset, etc.
Definition 1. A **lattice** is a mathematical structure $F = < F, \leq, \vee, \wedge, 0_F, 1_F >$, where F is a partially ordered set by the relation $\leq$, with the largest element 1_F, a smallest element 0_F, and $\vee$, $\wedge$ are internal composition laws of sup (or supremum), and inf (or infimum).
In many situations F is the Cartesian product of several lattices $F_j = <F_j, \leq_j, \vee_j, \wedge_j, 0_{Fj}, 1_{Fj}>$, for $j \in J = \{1,..., n\}$. We write this $F=F_1$ x...x F x...x F_n.
The relation $\leq$ on F is defined by $z = (z_1, ...,z_j, ...,z_n) \leq t = (t_1, ...,t_j, ...,t_n)$ iff $z_j \leq_j t_j$ for each j of J . We note $z \vee t = (..., z_j \vee_j t_j,....)$, $z \wedge t = (..., z_j \wedge_j t_j, ...)$, $O_F = (....,O_{Fj}, ...)$, $1_F = (..., 1_{Fj},…)$. (For standard Galois lattices, we have for each j: $F_j = \{0, 1\}$, $0 < 1$, $0 \vee_j 0 = 0$, $0 \vee_j 1 = 1 \vee_j 0 = 1 \vee_j = 1$, $0 \wedge_j 0 = 0 \wedge_j 1 = 1 \wedge_j 0 = 0$, $1 \wedge_j 1 = 1$. So $O_F = (...., 0, ...)$, and $1_F = (..., 1,...)$).
Definition 2. General contexts and descriptions:
Let m be a finite positive integer, $I = \{1,..., m\} = [1,…,m]$, and F any lattice. Let $d:I \rightarrow F$ be any mapping from I to F. A context C is defined as an array with rows d (i), i = 1, ..., m.
Formalism: The context provides, for each individual or object i of I, its description $d(i)=(d_1(i), d_2(i),…, d_j(i),..., d_n(i)) \in F = F_1$ x … x F_n, according to the attributes or properties $j \in J$. (In the standard case, $d_j(i)=1$ means that i has property j, and $d_j(i) = 0$ means that i has not the property j). So, in the general case, C is an m x n array of elements d_j (i) of F, and for each individual i and each property j, d_j (i) is the value of this property for i.
Definition 3. Galois connection:
Let $C = <I, F, d>$ be a context. We define $E = P(I)$, $|E|=2^I$ and $f: E \rightarrow F$ as follows: for each subset X of I, $f(X) = \wedge d(i)$: $i \in X\}$ if $X \neq \emptyset$, and $f(\emptyset) = 1_F$.
So, f(X) is the infimum of the descriptions d(i) of elements i of X. And in the standard case f(X) is an element $z = (z_1,..., z_j, ...)$ of F, and $z_j = \wedge_j \{d_j(i) : i \in X\} = 1$, iff $d_j(i)=1$, for each i of X. This means that $z_j = 1$ iff j is a property which belongs to all i in X. For this reason, we call f (X) the intent of X.

Remark:
For each i of I, we have f({i}) = d(i), and f is decreasing (if $X \subseteq X' \subseteq I$ then $f(X') \leq f(X)$).
We define g: $F \rightarrow E$ by $g(z) = \{i \in I: z \leq d(i)\}$, for each z of F.
We say that g(z) is the extent of z. (In standard case, g(z) is the set of all individuals who have all properties of z, $z_j = 1$).
We can see that g is also a decreasing mapping.
The ordered pair (f, g) is called a Galois connection. From it we derive two other mappings:
h : $P(I) \rightarrow P(I)$, by $h = g \circ f$, and $k = F \rightarrow F$ by $k = f \circ g$.
So, for each subset X of I, we have $h(X) = g(f(X)) = \{i \in I: f(X) \leq d(i)\}$, and for each z of F, we have $k(z) = f(g(z)) = \wedge\{d(i): i \in g(z)\}$.
We can see that h and k are closure operators. This means that each of them is an increasing, extensive, and idempotent operator. More explicitly, for each X, X' of E, and z, z' of F:

- $X \subseteq X'$ implies that $h(X) \subseteq h(X')$, $z \leq z'$ implies that $k(z) \leq k(z')$;
- $X \subseteq h(X)$, $z \leq k(z)$;
- $h(h(X)) = h(X)$, $k(k(z)) = k(z)$.

Any subset X of I such that X = h(X) is called a I-closed set, and each z of F such that z = k(z) is called F-closed element.
Let us define $H = \{X \subset I: h(X) = X\}$ the set of all closed subsets of I, and $K = \{z \in F: z = k(z)\}$, the set of all closed elements of F.
One can proof that there is a bijective mapping between H and K. The ordered pairs $(X, z) \in H \times K$ such that f(X) = z, and therefore such that g(z) = X, are called the concepts associated with the context C.
The set of all such concepts constitutes the Galois lattice GL(C) associated with this context C. (the order relation on GL(C) is defined by $(X, z) \leq (X, z')$ iff $X \subseteq X'$ and $z' \leq z$.)

3 ELL algorithm

This algorithm was proposed in [6]. We have subsequently improved it and compared to some other algorithms [2]. This comparison has shown better performance of ELL. This motivated our choice for this algorithm as a single-set G^2L computation.

Main idea:
For two disjoint subsets X_0 and K of the set of objects I, let ELL (X_0, K) denote a procedure which lists all the closed sets of I obtained by extending X_0 with some elements of K.
In other words, ELL (X_0, K) lists all the closed sets, which strictly contain X_0 and are contained in $X_0 \cup K$. Obviously, ELL (Ø, I) lists all the non-empty closed sets of I. ELL (X, K) proceeds by dichotomy as follows:

```
If (K ≠ Ø)
   Choose an element i0 ∈ K
   Find the closed sets which contain i0
  Find the closed sets which do not contain i0
Endif
```

The key point of the proposed algorithm is that the search time of such closed sets is considerably reduced by using the following proposition.

Proposition:

Let X_0 and $K \neq \emptyset$ be two disjoint subsets of I. Let $i_0 \in K$.

We have: $h(X_0 \cup \{i_0\}) = X_0 \cup A$ where $A = \{i \in I \backslash X_0 : f(X_0) \wedge f(i_0) \leq f(i)\}$

If a closed set contains X_0 and i_0, then it also contains A. Hence, if $A \subseteq K$ then $X_0 \cup A$ is the smallest closed set containing X_0 and i_0 and contained in $X_0 \cup K$.

If a closed set contains X_0 and does not contain i_0, then it also does not contain any element of the set: $R = \{i \in K : f(X_0) \wedge f(i) \leq f(i_0)\}$.

A (vs. R) is used for Attraction (vs. Rejection). The proof of this proposition is in [6].

The following pseudo-code is a recursive version of the algorithm.

```
Procedure ELL (X0, K)
GL = Ø // GL is a list of concepts
Var i0: element of I, z, z0: elements of F; X, A, R: subsets of I;
begin
  z0 = f(X0);
 if K ≠ Ø then
  begin
    Choose an element i0 of K;
    z = z0 ∧ f(i0); A = {i ∈ I\X0: z ≤ f(i)};
    if A ⊆ K then
    begin
      X = X0 U A; insert node (X, z) in GL;
     ELL (X, K\A);
    end;
   R = {i ∈ K: z0 ∧ f(i) ≤ f(i0)}; ELL (X0, K\R);
 end
end
```

The procedure ELL (Ø, I) starts with any $i_0 \in I$ (I is non empty). Then it determines the set A. Observing that $i_0 \in A$, we have the strict inclusions $\emptyset \subset X$ and $I \backslash A \subset I$. Hence if $A \subseteq K$, ELL (X, I\A) will run with a strictly smaller second parameter. Since $i_0 \in R$, we see that the same holds for ELL (X, I\R).

More generally ELL (X, K\A) and ELL (X_0, I\R) run with a strictly smaller second parameter than that of ELL (X, K). Since I is finite, this parameter which is a subset of I, will reach the void set and the procedure will terminate.

This algorithm lists all closed sets without duplicates. Let us show that each closed set F occurs exactly once.

Starting with GL = Ø, X_0 = Ø and K = I, let $i_0 \in I$ be fixed and consider two cases: $i_0 \in F$ and $i_0 \notin F$. If $i_0 \notin F$ then

- Either F is the smallest closed set which contains i_0. Then according to the previous proposition (a), $F=X=X_0 \cup A$ and F will be listed by ELL (X_0, K),
- Or F is not the smallest closed set which contains i0. In this case $X \not\subset F$ and F will be listed by ELL (X, K\A).

If $i_0 \notin F$, then according to the previous proposition (c), F will be listed by ELL(X_0,K\R). Since each insert only concerns the unique smallest closed set containing X_0 and i_0, we see that F occurs exactly once.
The only case not treated by the algorithm is whether the empty set is closed or not. The algorithm yields all the nodes (X, z) of the GL such that $X \neq Ø$. Since $f(Ø) = 1_F$ and $h(Ø) = \{i \in I: 1_F \leq f(i)\} = \{i \in I : f(i) = 1_F\}$, Ø is closed iff there is no $i \in I$ such that $f(i) = 1_F$.

4 Scalable Distributed G^{2}L calculus

We proposed a first solution in [1] which describes a way to parallelize the large contexts, by sharing a context C into sub-contexts depending on its rows or columns. Galois lattices associated with these sub-contexts are built, with the ELL, in different computers and then the global lattice is determined from these lattices. The algorithms used for the build of this global lattice are detailed in [1]. This solution was successfully tested. The study has shown that the time complexity appeared excessive for larger sets we have wished.
We have designed therefore a new solution to parallelize the work. It is based on a new algorithm, termed SD-ELL. The key idea is a recursive application of ELL.

4.1 SD-ELL algorithm

SD-ELL is composed of two algorithms. The first one computes the sets of objects K_i (i={1,…, n}, n is the number of the computed sets K_i) and the second one is duplicated on different machines to compute all the closed sets corresponding to each K_i.
The algorithm presented in the following table permits computing the sets K_i. It proceeds as follows:

- Choose an element $i_0 \in K$
- Compute the sets A and R
- Verify if $A \subseteq K$ then (X, Z) is a closed itemset and $K_i = K\backslash A$
- $K_{i+1} = K\backslash R$

The previous steps are reused with the parameters X_i and K_i. The process of decomposing K_i is stopped if $|K_i|$ is smaller than a fixed threshold or the total number of machine, in which the closed itemsets corresponding to each K_i are computed, is used.

Procedure KCompute(X_0,K_0,z_0)
LF = Ø; List of (*K*, *X*, *z)*
t = 1, *q* = 1, X_t = Ø, K_t = *I*, $z_t = 1_F$.
while ((*t* ≤ *q*) and ((*q* - *t* + 1)≤site number))
begin
$X_0 = X_t$; $K_0 = K_t$; $z_0=z_t$
Choose an element i_0 of *K*
$z = z_0 \wedge f(i_0)$;
$A = \{i \in I \backslash X_0: z \leq f(i)\}$;
$R = \{i \in K: z_0 \wedge f(i) \leq f(i_0)\}$;
if $A \subseteq K_0$ **then**
begin
$X = X_0 \cup A$; z = $z_0 \wedge f(i_0)$; $K = K_0 \backslash A$;
$LF = LF \cup (X, z)$
$q=q+1$; $X_q = X$; $K_q = K$; $z_q = z$;
end
$q = q + 1$; $X_q = X_0$; $K_q = K_0 \backslash R$; $z_q = z_0$; $t = t + 1$;
end

The second algorithm constituting SD-ELL is the iterative version of ELL (I-ELL) duplicated on different machines. This version is detailed in [6] and [2].
Let us use the context C from the example described below to illustrate how these two algorithms work.

O\A	a	b	c	d	e	f	g	h
1	12	16	6	10	12	13	12	6
2	12	12	8	12	11	13	11	8
3	10	13	16	14	11	8	12	10
4	13	14	11	10	13	13	12	14
5	17	10	10	14	13	10	14	12
6	0	14	3	8	4	13	11	10

For this example the threshold is 4 and the number of machine is 4.
The dispatcher executes the first algorithm (KCompute). KCompute starts with the parameters X_0=Ø and K= I ={1,2,3,4,5,6}. An element i0 is chosen such as i_0=1 and then the sets A, K1, R, and K2 are computed.

- z = $z_0 \wedge$ f (i_0) = {12,16,6,10,12,13,12,6}, A = {i ∈ I \ X_0: z ≤ f (i)}={1}. Then A⊆K:
  - $X_1=X_0 \cup A=\{1\}$ and $C_2(\{1\};\{12,16,6,10,12,13,12,6\})$ is a concept.
  - K_1 = K\A = {2,3,4,5,6}; $|K_1| > 4$, K1 must then be decomposed.
- R = {i ∈ K: $z_0 \wedge$ f (i) ≤ f (i_0)} = {1}⇒ K_2 = K\R = {2,3,4,5,6}; $|K_2| > 4$, K_2 must also be decomposed.

The previous steps are reused with the parameters (X_1 = {1}, K_1) and (X_0=Ø, K_2).

Decomposition of K_1: Choosing $i_0 \in K_1$ such as $i_0 = 2$.

• $z = z_0 \wedge f(i_0) = \{12,12,6,10,11,13,11,6\}$, $A = \{i \in I \setminus X: z \leq f(i)\} = \{2,4\}$. Then $A \subseteq K_1$:

- $X_3 = X_1 \cup A = \{1,2,4\}$ and C_3 ($\{1,2,4\};\{12,12,6,10,11,13,11,6\}$) is a concept.
- $K_3 = K_1 \setminus A = \{3,5,6\}$ ($|K_3| < 4$)

• $R = \{i \in K_1: z_0 \wedge f(i) \leq f(i_0)\} = \{2\} \Rightarrow K_4 = K_1 \setminus R = \{6,3,4,5\}$ ($|K_4| < 4$) with $X_4 = \{1\}$.

Decomposition of K_2: Choosing $i_0 \in K_2$ such as $i_0 = 2$.

• $z = z_0 \wedge f(i_0) = \{12,12,8,12,11,13,11,8\}$, $A = \{i \in I \setminus X: z \leq f(i)\} = \{2\}$. Then $A \subseteq K_2$:

- $X_5 = X_0 \cup A = \{2\}$ and $C_4(\{2\};\{12,12,8,12,11,13,11,8\})$ is a concept.
- $K_5 = K_2 \setminus A = \{3,4,5,6\}$ ($|K_5| < 4$)

• $R = \{i \in K_2: z_0 \wedge f(i) \leq f(i_0)\} = \{2\} \Rightarrow K_6 = K_2 \setminus R = \{6,3,4,5\}$ ($|K_6| < 4$) with $X_6 = \emptyset$.

Then, the dispatcher sends to the applications the parameters (K_3,X_3) (K_4,X_4) (K_5,X_5) (K_6,X_6). In parallel, the applications, described in the subsection 4.2, execute the I-ELL.

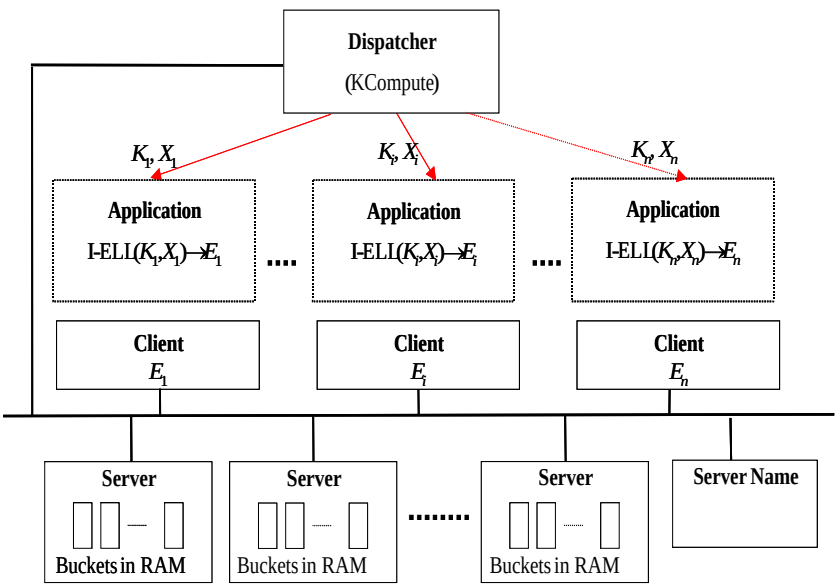
The following sets E_i are the sets of concepts generated by I-ELL(K_{i+2}, X_{i+2}) in the applications A_i.

The set of concepts E_1 generated by I-ELL (K_3,X_3). C_5({1,2,4,3};{10,12,6,10,11,8,11,6}) C_6({1,2,4,3,5};{10,10,6,10,11,8,11,6}) C_7 ({1,2,4,3,5,6};{0,10,3,8,4,8,11,6}) C_8({1,2,4,3,6};{0,12,3,8,4,8,11,6}) C_9 ({1,2,4,5};{12,10,6,10,11,10,11,6}) C_{10} ({1,2,4,5,6};{0,10,3,8,4,10,11,6}) C_{11} ({1,2,4,6} ;{0,12,3,8,4,13,11,6})	The set of concepts E_2 generated by I-ELL(K_4, X_4). C_{12}({1,6,4};{0,14,3,8,4,13,11,6}) C_{13}({1,6,4,3};{0,13,3,8,4,8,11,6}) C_{14}({1,3,4};{10,13,6,10,11,8,12,6}) C_{15} ({1,3,4,5};{10,10,6,10,11,8,12,6}) C_{16} ({1,5,4};{12,10,6,10,12,10,12,6}) C_{17}({1,4};{12,14,6,10,12,13,12,6})
The set of concepts E_3 generated by I-ELL(K_5, X_5). C_{18}({2,3};{10,12,8,12,11,8,11,8}) C_{19}({2,3,4};{10,12,8,10,11,8,11,8}) C_{20}({2,3,4,5};{10,10,8,10,11,8,11,8}) C_{21}({2,4,3,5,6};{0,10,3,8,4,8,11,8}) C_{22}({2,4,3,6}; {0,12,3,8,4,8,11,8}) C_{23}({2,3,5};{10,10,8,12,11,8,11,8}) C_{24}({2,6,4};{0,12,3,8,4,13,11,8}) C_{25}({2,6,4,5};{0,10,3,8,4,10,11,8}) C_{26}({2,5};{12,10,8,12,11,10,11,8}) C_{27}({2,5,4};{12,10,8,10,11,10,11,8}) C_{28}({2,4};{12,12,8,10,11,13,11,8})	The set of concepts E_4 generated by I-ELL(K_6, X_6). C_{29}({6,4);{0,14,3,8,4,13,11,10}) C_{30}{6,4,3}; {0,13,3,8,4,8,11,10}) C_{31}({6,4,3,5};{0,10,3,8,4,8,11,10}) C_{32}({6,4,5};{0,10,3,8,4,10,11,10}) C_{33}({3};{10,13,16,14,11,8,12,10}) C_{34}({3,4};{10,13,11,10,11,8,12,10}) C_{35}({3,4,5};{10,10,10,10,11,8,12,10}) C_{36}({3,5};{10,10,10,14,11,8,12,10}) C_{37}({4};{13,14,11,10,13,13,12,14}) C_{38}({4,5};{13,10,10,10,13,10,12,12}) C_{39} ({5};{17,10,10,14,13,10,14,12})

The union of E_1, E_2, E_3, E_4, and the concepts calculated in the dispatcher is the set of all concepts corresponding to the context C.

4.2 The Architecture

A major problem for the final efficiency of SD-ELL is the storage of the collection of the closed produced, presumably very large and of unknown size. Our tool is a Scalable Distributed Data Structure (SDDS). An SDDS dynamically distributes over the (server) nodes whose number dynamically scales with the file growth. The application accesses the file through the client nodes making the data distribution transparent. For scalability, the data address calculations do not involve any centralized directory. The data are also basically stored in the distributed RAM, for faster access than to the traditional disk-based structures [10]. All these properties are prime importance for our goal.
Our first architecture for SD-ELL, illustrated below, calculus is to add to the SDDS structure of clients and servers, a dispatcher and our applications at each available client site. The dispatcher executes the first algorithm of SD-ELL. It then sends to the applications the parameters K_i and X_i. In parallel, on each client the iterative version is executed with its parameters. We obtain the closed itemsets that after the calculus partition the Galois lattice. The closed sets are sent by the clients to the servers, where they are saved.



The dispatcher is a centralized component. To offset the potential drawbacks of this approach, we have designed also a more distributive solution. The dispatcher calculates only the set K_1 and K_2 ($K_1=K_0\backslash A$ and $K_2 =K_0\backslash R$) and sends them to two applications. These applications become dispatchers on there turn. They send to other

applications the parameters K_i and X_i if K_i is greater than a fixed threshold and there are available machines.
We will first implement and measure the 1st solution. Later we will also design the end one and compare the related trade-offs.
Our preliminary experimentations has shown that the processing time of the distributed SD-ELL algorithm out performs the processing time of the ELL algorithm. Besides, increasing the number of processors improves the processing time.

5 Related work

In [12], Ndoundam and al. proposed a methodology which permits implementing a parallel version of Bordat [14] through the parallelization of nested loops. Nevertheless, this parallelization is incompleted as proved by [13] how propose a better fine-grain parallelization.
By another way, [15] present a parallel version of Ganter algorithm [7] for large context which relies on a partitioning of the search space. Let us note that all these previous works do not provide effective implementation of the parallel algorithms. Only [11] propose a validation of their theoretical model through an effective implementation.

6 Conclusion

In this paper, we propose a new algorithm (SD-ELL) which is a distributed version of ELL, in order to deal with large databases. The architecture used for this distribution allows to share the computation of closed sets of large databases on the different applications and to store them in the server nodes. The storage of the great number of closed sets is successful since the number of server nodes dynamically scales with the growth of the closed itemsets file. Our ongoing research consists in continuing the evaluating of the prototype performance measurements. A possible future research direction consists in determining the links between the concepts and then comparing our results with the results of [11].

References

1. Baklouti. F, Lévy. G. Parallel algorithms for general Galois lattices building. Proc. WDAS 2003.
2. Baklouti. F, Lévy. G. A fast and general algorithm for Galois lattices building. Journal of Symbolic Data Analysis. Vol 3. pp. 19-31.
3. Birkhoff. G. Lattice Theory. American Mathematical Society, Providence, RI, 3rd edition. 1967.
4. Bordat. J. Calcul pratique du treillis de galois d'une correspondance, Mathématique, Informatique et Sciences Humaines 24(94), 31. 1986.

5. Diday. E, Emilion. R. Treillis de Galois maximaux et capacités de Choquet. Cahier de Recherche de l'Acadèmie des Sciences. Paris, t.325, Série I, p.261-266, 1997.
6. Emilion. R. Lambert. G, Lévy. G, Algorithmes pour les treillis de Galois, Indo-French Workshop, University Paris IX-Dauphine. 1997.
7. Ganter. B. Two basic algorithms in concept analysis. Preprint 831, Technische Hochschule Darmstadt 1984.
8. Godin. R., Mineau. G. and al. Méthodes de classification conceptuelle bases sur les treillis de Galois et application. Revue d'intelligence artificielle pp 105-137. 1995.
9. Ganter. B, Wille. R. Formal Concept Analysis. Mathematical Foundations, Springer. 1999.
10. Litwin. W., Neimat, M-A., Schneider. D. A Scalable Distributed Data Structures. ACM Transactions on Database Systems (ACM-TODS), Dec. 1996.
11. Valtchev, P., Djoufak Kengue, J,F., Djamegni C,T., A parallel algorithm for lattice construction, in the proceding of the third ICFCA, February 2005, Lens, Fance. pp 249-264.
12. Ndoundam, R., Njiwoua, P., Mephu Nguifo, E. Une etude comparative de la parallelisation d'algorithmes de construction de treillis de Galois. Atelier francophone de la plate forme de l'AFIA. Usage des treillis de Galois pour l'intelligence artificielle, ESIEA Recherche (2003).
13. Njiwoua, P., Mephu Nguifo, E. A parallel algorithm to build concept lattice. In proceedings of the fourth Groningen Int. Information Tech. Conf. for students (1997).
14. Bordat, J.P. Calcul pratique du treillis de Galois d'une correspondance. Mathématiques et Sciences Humaines, Vol. 96 (1986) 31-47.
15. Fu, H., Mephu Nguifo, E. A parallel algorithm to generate formal concepts for large data. In proceedings of the second international conference on formal concept analysis (ICFCA04), springer LNCS, Sydney Australia (2004).
16. Diday. E, Emilion. R. Treillis de Galois maximaux et capacités de Choquet. Cahier de Recherche de l'Acadèmie des Sciences. Paris, t.325, Série I, p.261-266, 1997.
17. Diday. E, Emilion. R. "Maximal and stochastic Galois lattices", Discrete Applied Mathematics, 127, (2003), 271-284.

Acknowledgements

We would like to express special thanks to Pr. Witold Litwin for his precious remarks and his help for using the SDDS.