A Lattice Algorithm for Data Mining

Huaiguo Fu, Engelbert Mephu Nguifo

CRIL-CNRS FRE2499, Université d'Artois-IUT de Lens
Rue de l'université SP 16, 62307 Lens cedex. France
{fu,mephu}@cril.univ-artois.fr

ABSTRACT. *Concept lattice is an effective tool and platform for data analysis and knowledge discovery such as classification or association rules mining. The lattice algorithm to build formal concepts and concept lattice plays an essential role in the application of concept lattice. In fact, more than ten algorithms for generating concept lattices were published. As real data sets for data mining are very large, concept lattice structure suffers from its complexity issues on such data. The efficiency and performance of concept lattices algorithms are very different from one to another.In order to increase the efficiency of concept lattice-based algorithms in data mining, it is necessary to make use of an efficient algorithm to build concept lattices.So we need to compare the existing lattice algorithms and develop more efficient algorithm. We implemented the four first algorithms in Java environment and compared these algorithms on about 30 datasets of the UCI repository that are well established to be used to compare ML algorithms. Preliminary results give preference to Ganter's algorithm, and then to Bordat's algorithm, nevertheless these algorithms still suffers when dealing with huge datasets. We analyzed the duality of the lattice-based algorithms. Furthermore, we propose a new efficient scalable lattice-based algorithm: ScalingNextClosure to decompose the search space of any huge data in some partitions, and then generate independently concepts in each partition. The experimental results show the efficiency of this algorithm.*

RÉSUMÉ.

KEYWORDS: *Concept lattice, data mining, lattice algorithm*

MOTS-CLÉS :

1. Introduction

Concept is an important and basic means of knowledge representation, since it represents abstraction and generalization of objects. A concept defines a subset of objects which shares some common attributes or properties. Concept lattice structure [BIR 67, BAR 70, GAN 99] has shown to be an effective tool for data analysis, knowledge discovery, and information retrieval, etc [Mep 02]. It shows how objects can be hierarchically grouped together according to their common attributes. Researchers of different domains study it in theory and application of data analysis and formal knowledge representation etc.

Several algorithms are proposed to build concepts or concept lattices of a context : Bordat [BOR 86], Ganter (NextClosure) [GAN 84], Chein [CHE 69], Norris [NOR 78], Godin [GOD 95], Nourine [NOU 99], Carpineto [CAR 96], and Valtchev [VAL 02], etc. Some algorithms can generate also diagram graphs of concept lattices. The performance of the lattice algorithm is very important for its application to data mining (DM). In fact real data sets for DM are very large, e.g. the customer data of a company. In the worst case, the generation of lattice nodes increases exponentially. The efficiency of concept lattice algorithms are different from one to another. So we need to compare the existing lattice algorithms with large data and make use of an efficient algorithm to satisfy the mining and learning task and to increase the efficiency of concept lattice-based algorithms in real applications.

Different works on comparison of lattice algorithms have been done. Guénoche [GU9̃0] reviewed four algorithms: Chein, Norris, Ganter and Bordat. This is the first review of lattice algorithms, he pointed out theoretical complexity, but there is no experimental test for these algorithms. Godin et al. [GOD 98] presented incremental algorithms for updating the concept lattice and corresponding graph. Results of empirical tests were given in order to compare the performance of the incremental algorithms to three other batch algorithms: Bordat, Ganter, Chein. The test data is small and randomly generated. Kuznetsov et al. [KUZ 02] compared, both theoretically and experimentally, performance of ten well-known algorithms for constructing concept lattices. The authors considered that Godin was suitable for small and sparse context, Bordat should be used for contexts of average density, and Norris, CBO and Ganter should be used for dense contexts. The algorithms were compared on different randomly generated contexts using the density/sparseness, and on one real dataset (SPECT heart database) of the UCI repository. The test data is small and randomly generated, only one real dataset is used.

If the experimental datasets are too small or random, it's not easy to appraise the performance of these algorithms for DM. So in order to analyze and compare concept lattices algorithms, we use a publicly available database [BLA 98] which are often used in order to compare machine learning (ML) algorithms. Even if it is not demonstrated that this database which contains more than forty datasets is representative of practical applications, it is well established that these testbeds should be used to measure efficiency issues of a new ML algorithm. So it's necessary to show how concept lattice algorithms fits in such data. Conclusions could help to build efficient ML algorithm based on concept lattice.

When generating concepts, lattice algorithm focusses on objects or attributes. So if the number of objects is greater than the number of attributes, it might be interesting to build the concept node based on the minimum number between objects and attributes [FU 03a, RIO 03]. We propose a new definition: dual algorithm, which consists of applying an algorithm to the same context by inverting rows and columns. The duality of lattice algorithm is considered in our comparison of lattice algorithms. The difference between algorithm and its dual algorithm is described.

We implemented the four first published algorithms (Chein, Norris, Ganter and Bordat) and their dual algorithms for generating concept lattices in Java environment. The other algorithms are very often extension of these 4 algorithms. We compared these algorithms on about 30 datasets of the UCI repository that are well established to be used to compare machine learning algorithms. We test also these algorithms in the worst case. Preliminary results give preference to Ganter's algorithm, and then to Bordat's algorithm.

Although the experimental comparisons of performance of existing algorithms show that NextClosure algorithm is the best for large and dense data [KUZ 02, FU 03a], it still takes expensive time cost to deal with huge data. So in this paper, we propose a new efficient lattice-based algorithm ScalingNextClosure that decomposes the search space of any huge data in some partitions, and then generates independently concepts or closed itemsets in each partition.

The new algorithm is a kind of decomposition algorithm of concept lattice. All existing decomposition algorithms [VAL 02] for generating concept lattices use an approach of context decomposition, that are different from ours. Our new algorithm uses a new method to decompose the search space. It can freely decompose the search space in any set of partitions if there are concepts in each partition, and then generate them independently in each partition. So this algorithm can be used to analyze huge data and to generate formal concepts. Moreover for this algorithm, each partition only shares the same source data (data context) with other partitions. ScalingNextClosure algorithm shows good scalability and can be easily used for parallel, distributed network and partial computing [FU 04].

The experimental comparison with NextClosure algorithm shows that our algorithm (only sequential computing) is better for large data. Our algorithm succeeds in computing some large data in worst case that are impossible to be computed with another algorithm.

The rest of this paper is organized as follows : we introduce the notion of concept lattice in section 2. In section 3, experimental comparisons of the four lattice algorithms are discussed. The new algorithm ScalingNextClosure will be presented in section 4. In section 5, the performance of the new algorithm will be shown. The paper ends with a short conclusion in section 6.

2. Concept lattice

The theoretical foundation of concept lattice relies on the mathematical lattice theory [BIR 67]. Concept lattice is used to represent the order relation of concepts.

Definition 2.1 *A* **context** *is defined by a triple $(G; M; R)$, where G and M are two sets, and R is a relation between G and M. The elements of G are called objects, while the elements of M are called attributes.*

For example, Figure 1 represents a context. $G(1, 2, 3, 4, 5, 6, 7, 8)$ is the object set, and $M(a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8)$ is the attribute set. The crosses in the table describe the relation R between G and M, which means that an object verifies an attribute.

	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8
1	×	×					×	
2	×	×					×	×
3	×	×	×				×	×
4	×		×				×	×
5	×	×		×		×		
6	×	×	×	×		×		
7	×		×	×	×			
8	×		×	×		×		

Figure 1. *An example of context (G, M, R).*

Definition 2.2 *Given a subset $A \subseteq G$ of objects from a context $(G; M; R)$, we define an operator that produces the set A' of their common attributes for every set $A \subseteq G$ of objects to know which attributes from M are common to all these objects:*

$A' := \{m \in M \mid gRm \text{ for all } g \in A\}$.

Dually, we define B for subset of attributes $B \subseteq M$, B' denotes the set consisting of those objects in G that have all the attributes from B:

$B' := \{g \in G \mid gRm \text{ for all } m \in B\}$.

These two operators are called the **Galois connection** *of $(G; M; R)$. These operators are used to determine a formal concept.*

So if B is an attribute subset, then B' is an object subset, and then $(B')'$ is an attribute subset. We have: $B \subseteq M \Rightarrow B'' \subseteq M$. Correspondingly, for object subset A, we have: $A \subseteq G \Rightarrow A'' \subseteq G$.

Definition 2.3 *A* **formal concept** *of the context (G, M, R) is a pair (A, B) with $A \subseteq G$, $B \subseteq M, A = B'$ and $B = A'$. A is called extent, B is called intent.*

Definition 2.4 *If* (A_1, B_1) *and* (A_2, B_2) *are concepts,* $A_1 \subseteq A_2$ *(or* $B_2 \subseteq B_1$*), then we say that there is a hierarchical order between* (A_1, B_1) *and* (A_2, B_2)*.*

All concepts with the hierarchical order of concepts form a complete lattice called **concept lattice***.*

3. Comparison of concept lattice algorithms

Concept lattice algorithm plays an essential role for the application of concept lattice. More than ten algorithms for generating concept lattices were published. Generally, concept lattice algorithms are divided into two types: batch algorithms and incremental algorithms. Batch algorithms construct completely the lattice from scratch when adding a new object or attribute, while incremental ones update lattice structure when adding a new object.

For example, algorithms of Bordat, Ganter, Chein, Lindig and Nourine are batch algorithms. There are three ways to generate concepts with batch algorithms :

– **Descending:** such as Bordat's algorithm, from the top concept, we build the maximal rectangles. The algorithm repeats the same process to generate the other subnodes.

– **Ascending:** We can generate concepts below, and then spread super-node, such as Chein algorithm.

– **Enumeration:** algorithm enumerates all the nodes of the lattice according to a certain order. For example, Ganter's algorithm uses lexicographical order to enumerate the nodes.

There are some incremental algorithms such as the algorithms of Norris, Godin, Capineto, Dowling and Valtchev. The idea of these algorithms is that the new object makes intersection with all the concepts in the lattice to update lattice structure.

The algorithms of Chein, Norris, Ganter and Bordat are four first published algorithms. They belong to 4 typical lattice algorithms (descending, ascending, enumeration batch algorithms and incremental algorithms). The other algorithms are very often improvements or extensions of one of these four algorithms. Therefore we select the four algorithms to compare and analyze them on different aspects.

3.1. *The algorithms principle*

3.1.1. *Chein's algorithm*

Chein's algorithm [CHE 69] builds concepts in a bottom-up manner. It repeats the following iterative method at every stage k.

For each object g_i, $(g_i,(g_i'))$ is considered as first layer L_1. L_k is the set of the rectangles of layer k. An arbitrary element of L_k is (G_i,G_i'). From L_k, we build the layer L_{k+1}. For every two elements of L_k : (G_i, G'_i) and (G_j, G'_j), if $G'_i \cap G'_j \notin L_{k+1}$, then $(G_i \cup G_j, G'_i \cap G'_j)$ is an element of L_{k+1}. Otherwise, merge all pairs that have the same $G'_i \cap G'_j$ as an element of L_{k+1}.

At the end we delete L_k's element whose attribute set is the same as L_{k+1}'s element.

3.1.2. *Norris' algorithm*

Norris' algorithm [NOR 78] is an incremental algorithm. For the context (G, M, R), when we add each objects g_k, the concepts set of this level L_K is generated from L_{K-1} in the same way. For the first object, L_1 contains only (g_1, g'_1).

Adding one object g_{k+1} to L_K, we can build L_{k+1}. $\forall(G_i,\, G'_i) \in L_K$. If $G'_i \subset g'_{k+1}$, then $(G_i \cup (g_{k+1}), G'_i) \in L_{k+1}$.

Otherwise, $(G_i, G'_i) \in L_{k+1}$, and furthermore we add $(G_i \cup (g_{k+1}),\, G'_i \cap (g'_{k+1}))$ to L_{k+1} if $(G_i,\, (g'_{k+1}) \cap G'_i)$ is maximum.

After examination of all the rectangles, if g'_{k+1} is maximum, we add the $(g_{k+1}$, $g'_{k+1})$ in L_{k+1}.

3.1.3. *Ganter's algorithm (NextClosure algorithm)*

The principle of NextClosure algorithm [GAN 84] uses the characteristic vector which represents arbitrary subsets A of M, to enumerate all concepts of $(G; M; R)$. Given $A \subseteq M$, $M = \{a_1, a_2, \ldots, a_i, \ldots, a_{m-1}, a_m\}$, $A \to A''$ is the closure operator. The lectically smallest attribute subset is $\emptyset''$. The NextClosure algorithm proved that if we know an arbitrary attribute subset A, the next concept (the smallest one of all concepts that is larger than A) with respect to the lexicographical order is $A \oplus a_i$, where $\oplus$ is defined by

$$A \oplus a_i = (A \cap (a_1, a_2, \ldots, a_{i-1}) \cup \{a_i\})''$$

$A \subseteq M$ and $a_i \in M$, a_i being the largest element of M with $A < A \oplus a_i$ by lexicographical order.

In other words, for $a_i \in M \backslash A$, from the largest element to smaller one of $M \backslash A$, we calculate $A \oplus a_i$, until we find the first time $A < A \oplus a_i$, then $A \oplus a_i$ is the next concept.

3.1.4. *Bordat's algorithm*

The Bordat's algorithm [BOR 86] searches all concepts hierarchically and builds the concept lattices (Hasse diagram). It uses a top-down strategy, and is a level-wise algorithm. Its principle is first to find all the maximal object subsets of G, then to build the corresponding concepts, and finally to find the maximal object subsets of the object subsets found above. So there are clear hierarchical relations within all concepts of a context, so that we can generate concept lattices.

Bordat's algorithm doesn't only generate all concepts but also it builds links between these nodes. This procedure increases the time cost. So it needs large memory.

3.2. *Dual algorithm*

Analyzing the four algorithms, we find that one algorithm can focuss on objects or attributes. The performances of an algorithm can be different according to the number of objects and attributes. So every lattice algorithm can be described or implemented by focussing either on objects or on attributes. We propose a new definition: dual algorithm.

Definition 3.1 *A* **dual algorithm** *of concept lattice is an algorithm which can be applied to the same context by focussing either on objects or on attributes.*

In other words: we can use the same algorithm from two directions (objects (set) or attributes (set)) to generate the concept lattice. Two dual algorithms are usually considered to be the same, and we can get the same concept lattice with two dual algorithms. In fact, the idea of the algorithms is the same, but the time cost of algorithm isn't frequently identical.

Proposition 3.1 *The time cost of a dual algorithm for a context is equivalent to the time cost of original algorithm for dual context.*

A dual context of a context is obtained by inverting rows and columns. This is also called transposed matrix or context [RIO 03].

3.3. *Experimental comparison*

The four algorithms and their corresponding dual ones are implemented in Java environment and are available through request. These algorithms are tested on a Pentium III 450, 128 MB RAM. In our experiment, we compared these algorithms on about 30 datasets of the UCI repository and on the worst cases.

3.3.1. *Test on ML benchmarks*

Benchmark databases

Real data for our experiment come from ML benchmarks: UCI repository. We have got about 30 databases to build binary contexts (see table 1). The biggest context has 67557 objects and 126 attributes. This is not as huge as on real databases. However it's larger than datasets used by Kuznetsov et al. [KUZ 02] and Godin et al. [GOD 98] in their experiments.

DataSet	ID	Objects	Attributes	Concepts
shuttle-landing-control	d03	15	24	52
adult-stretch	d01	20	10	89
lenses	d02	24	12	128
zoo	d07	101	28	377
hayes-roth	d06	132	18	380
servo	d09	167	19	432
SPECT_train	d04	80	23	909
post-operative	d05	90	25	1521
balance-scale	d18	625	23	2104
flare1	d17	323	32	2608
flare2	d21	1066	32	2987
soybean-small	d10	47	79	3253
monks-3	d14	432	19	3959
monks-1	d16	432	19	4463
monks-2	d15	432	19	5427
car	d22	1728	21	7999
breast-cancer-wisconsin	d25	699	110	9860
house-votes-84	d13	435	18	10642
SPECT_test	d11	187	23	14532
SPECT_two	d30	267	23	21548
audiology.standardized	d08	26	110	30401
tic-tac-toe	d20	958	29	59503
nursery	d27	12960	31	147577
lung-cancer	d12	32	228	186092
agaricus-lepiota	d28	8124	124	227594
promoters	d19	106	228	304385
soybean-large	d23	307	133	806030
dermatogogy	d24	366	130	1484088
kr-vs-kp	d26	3196	75	/
connect-4	d29	67557	126	/

Table 1. *The datasets of UCI repository ordered by the number of concepts. / means that the programs fail to generate all concepts.*

These datasets are ordered by the number of concepts. For two datasets (kr-vs-kp and connect-4), we didn't get the number of concepts with these algorithms in our computer, as they fail due to the lack of memory.

Running time of the 4 first algorithms

We tested every context with the four first algorithms. Figure 2 shows the running time results. Analyzing the experimental results, Ganter and Bordat algorithms are faster than others. Bordat's algorithm not only generates the nodes of the lattice but also it builds links between these nodes. So if we want really to compare the three others to Bordat's algorithm, it would be necessary to build their links between nodes.

Running time of the 4 dual algorithms

We consider that the performance is different between one algorithm and its dual algorithm. So we implement each algorithm and its dual algorithm to focus respectively on objects or attributes. The experimental results (see table 2) show that the performance of two dual algorithms are very different. For example, we have tested Ganter's algorithm and its dual algorithm for the dataset Flare2, and time cost can

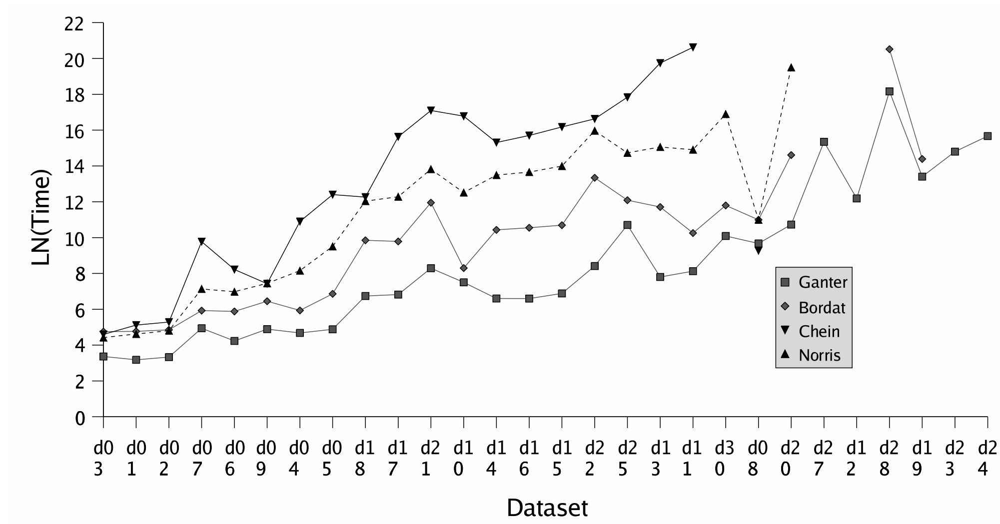


Figure 2. *Performance (in ms) of the* 4 *fi rst lattice algorithms on UCI datasets.*

be 100 times different. So the difference between algorithm and its dual algorithm is marked, to show that we must consider duality when comparing lattice algorithms.

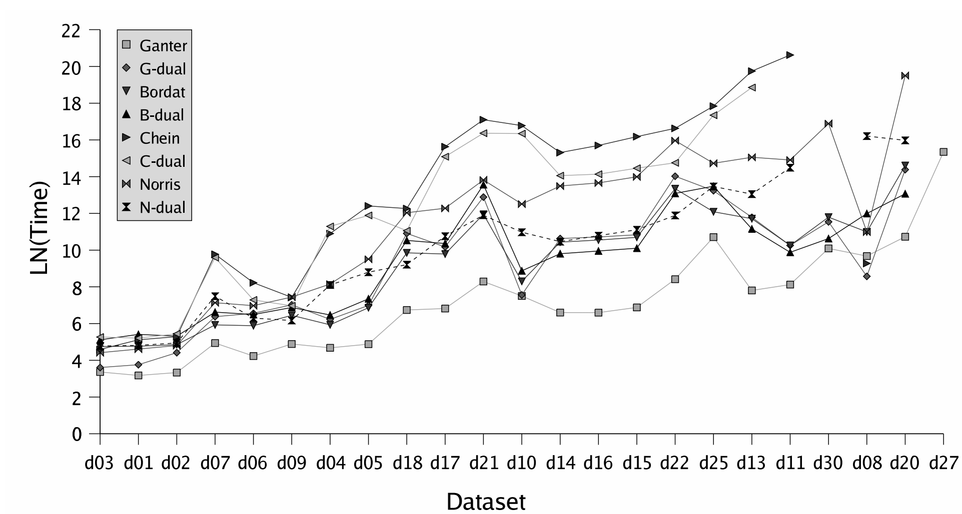


Figure 3. *Performance of lattice algorithms and their dual algorithms.*

ID	Ganter	G-dual	Bordat	B-dual	Chein	C-dual	Norris	N-dual
d03	29	37	116	167	99	194	83	119
d01	24	43	118	224	167	183	101	123
d02	28	83	128	204	198	232	122	140
d07	140	589	376	754	17607	14652	1255	1787
d06	69	707	359	660	3724	1462	1070	543
d09	133	1120	633	968	1681	1061	1713	478
d04	108	507	378	646	54283	80217	3467	3297
d05	132	1084	957	1554	243564	146154	13457	6620
d18	845	54825	18986	37388	211849	63231	168423	9986
d17	916	26149	17798	31592	6146858	3562383	214944	46890
d21	4000	395911	154984	778299	26633173	12807020	1002709	147099
d10	1819	1924	4027	7143	19392256	12515300	271203	58280
d14	737	41453	34005	18126	4457627	1272022	721953	34865
d16	734	45055	38153	20997	6583877	1376935	853978	48280
d15	975	50789	44152	24404	10623849	1904268	1190532	66662
d22	4516	1229305	623112	483612	16677919	2555755	8566689	145633
d25	44748	560389	177929	713596	55932215	34140525	2485251	703593
d13	2450	131862	121844	69740	375882938	154525507	3461533	463109
d11	3382	28432	28459	19540	/	/	1982204	1059784
d30	24144	102877	132799	41304	/	/	21695260	/
d08	15922	5288	59384	161548	10672	/	59192	10914983
d20	45681	1767983	2211559	472985	/	/	294433666	8667683
d27	4627030	/	/	/	/	/	/	/
d12	196487	75805	/	496263	/	/	/	/
d28	77183183	/	/	/	/	/	/	/
d19	663052	552271	1774334	1469483	/	/	/	/
d23	2676454	14959171	/	/	/	/	/	/
d24	6367387	/	/	/	/	/	/	/
d26	/	/	/	/	/	/	/	/
d29	/	/	/	/	/	/	/	/

Table 2. *The results of running time (in milliseconds) of lattice algorithms for real data. / means that the programs fails to generate all concepts.*

Figure 3 shows performance of the four algorithms and their dual algorithms. We can see that Ganter's algorithm runs faster than others. Figure 4 shows an important conclusion: Ganter's algorithm has the best performance when it focusses on the smallest number of objects or attributes. For example, for dataset d08, it has 26 objects and 110 attributes, the number of objects is smaller than attributes, so dual algorithm that focusses on objects is faster than that focussing on attributes. With the real database, the number of attributes is often smaller than that of objects. Ganter's algorithm works faster than others in this case. Ganter's algorithm should search all closures using the smallest number between attributes or objects. This is the consequence of Ganter's algorithm since it explores almost all the subsets of the set of attributes or objects.

This is not the case with the three other algorithms. Norris' algorithm and its dual algorithm have the most difference. But Bordat's algorithm have little difference with its dual algorithm. It is not possible to infer from the analysis of the code of the 3 other algorithms that they should be used by focussing on the smallest number of attributes or objects. And the experiment seems to confirm that.

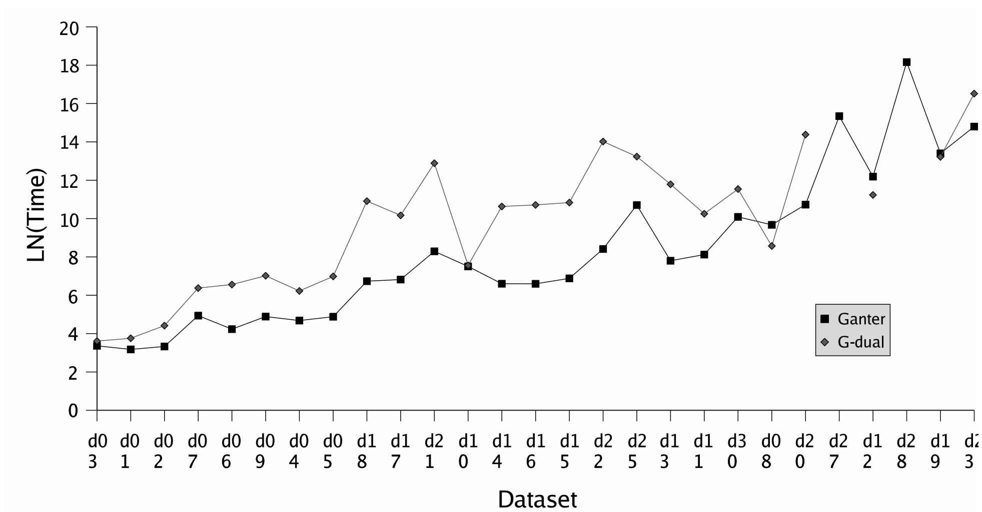


Figure 4. *Performance of comparison on UCI datasets with Ganter algorithms.*

	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8
1		×	×	×	×	×	×	×
2	×		×	×	×	×	×	×
3	×	×		×	×	×	×	×
4	×	×	×		×	×	×	×
5	×	×	×	×		×	×	×
6	×	×	×	×	×		×	×
7	×	×	×	×	×	×		×
8	×	×	×	×	×	×	×	

Figure 5. *An example of worst case data context.*

3.3.2. *Test in the worst case*

Definition 3.2 *A context in the* **worst case** *is the case where the sizes of G and M are equal to n, and each attribute is verified by $n-1$ different objects, each object possesses $n-1$ different attributes.*

Figure 5 is an example of worst case with the size of data context equal to 8.

We have tested four algorithms in the worst case. This particular case generates with context of size n (number of lines and number of columns): 2^n nodes for concept lattice. The results (see figure 6) show that Ganter's algorithm is the best in worst case. It succeeds in computing some large data that were impossible to be computed with other algorithms. For example: the worst case with 20 attributes (2^{20} concepts) is very hard to compute with other algorithms, but Ganter's algorithm can build the concept lattice for this context. However each algorithm fails in building lattice nodes for context with more than 22 attributes.

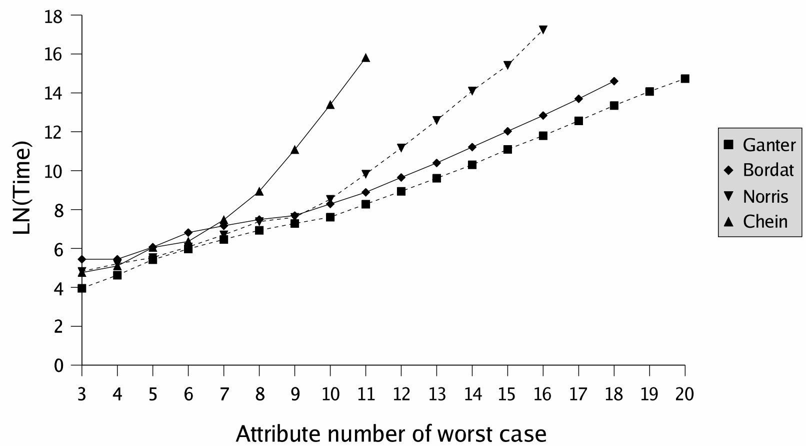


Figure 6. *Performance of comparison on worst case data.*

Although the experimental results show that NextClosure algorithm is the best for large and dense data, it still takes expensive time cost to deal with huge data. Large data is one big challenge for data mining algorithms. The one of the most effective solution is divide-and-conquer. So we try to decompose the search space of large data in some partitions, and then generates independently concepts in each partition.

4. ScalingNextClosure algorithm

Analyzing all concepts of a data context and their search space, we define the Ordered data context, and analyze the property of the ordered data context in order to decompose the search space.

Definition 4.1 *A data context is called* **ordered data context** *if the attributes are ordered by number of objects of each attribute from the smallest to the biggest one (In other words: the smallest attribute is in fi rst column, and the biggest one is in the last column), and the attributes with the same objects are merged as one attribute.*

Proposition 4.1 *An ordered data context has the same concept lattice as the data context.*

Proof : By the definition of ordered data context, G, M and R aren't changed, so the data context does not change by the preprocessing step. There is a unique concept lattice for any given context[GAN 99].

It's the precondition of ScalingNextClosure algorithm to transform a data context to the ordered data context. It is easy to generate the ordered data context.

We propose a new method to divide the search space of concepts into partitions. For each partition, we can find all concepts in it. This is the principle of our ScalingNextClosure algorithm. In following section, we will present the main idea of ScalingNextClosure algorithm and explain why and how we can divide the search space into partitions and then generate concepts in each partition.

4.1. *The search space for concepts*

Intent and extent of a concept are bijection, so we can only study the search space of intent or extent of concepts instead of search space for concepts. In fact, any concept intent is a attribute subset of M, so all subsets of M are elements of the search space. So the size of search space for enumeration of all concept intents is 2^m. This search space can be considered as the fold of some attribute subsets of M. For example, we consider that the search space is formed by $folds$, where $fold_i$ is all subsets of $\{a_i$, …, a_{m-3}, a_{m-2}, a_{m-1}, $a_m\}$ that include a_i. Each $fold$ is a search sub-space. The whole search space will be decomposed according to the situations of such $folds$. Here we define Folding set and Folding search sub-space in order to decompose the search space.

Definition 4.2 *The attribute set M of a data context $(G;M;R)$ is $\{a_1$, …, a_i, …, a_{m-2}, a_{m-1}, $a_m\}$, an attribute $a_i \in M$, the set F_{a_i} is called* **folding set** *of a_i, where*

$F_{a_i} := \{a_j \in M|$ *for all* $a_j \in M, i < j \leq m\}$

In other words, the folding set of a_i is the set of $\{a_{i+1}, a_{i+2}, \ldots, a_{m-1}, a_m\}$.

For example, the folding set of a_m is $\emptyset$. For attribute a_{m-3}, its folding set is $\{a_{m-2}, a_{m-1}, a_m\}$.

Definition 4.3 *An attribute joins respectively with all subsets of its folding set to generate the new attribute subsets, these new attribute subsets form a search sub-space of concepts that is called* **folding search sub-space of an attribute(F3S)**.

For example, F3S of a_{m-1} is: $\{a_{m-1}\}$; $\{a_{m-1}$,$a_m\}$. F3S of a_i is: all subsets of $\{a_i$,…,a_{m-2},a_{m-1},$a_m\}$ that include a_i.

Proposition 4.2 *For a data context $(G;M;R)$, $\forall a_i \in M$, the number of attributes of M is m, if the number of objects of attribute a_i is n, then the folding search sub-space (for concepts) of a_i is the minimum of 2^n and 2^{m-i}.*

Proof : $\forall a_i \in M$. The folding set of a_i is $\{a_{i+1}, a_{i+2}, \ldots, a_{m-2}, a_{m-1}, a_m\}$, it has $m-i$ attributes. So the size of subset of the folding set of a_i is 2^{m-i}, a_i can be assembled with 2^{m-i} attribute subsets to form new attribute subsets.

On the other hand, if the number of objects of attribute a_i is n, we have 2^n object subsets corresponding to a_i. For any concept, it's a bijection between concept and corresponding object set. So there are at most 2^n concepts that include attribute a_i. Thus the folding search sub-space of a_i is the minimum of 2^n and 2^{m-i}.

According to this proposition, we order the data context with the number of objects of each attribute. In practice, this arrangement can remarkably reduce the search space for real data.

In the definition of ordered data context, we need to merge the attributes with exactly the same objects as one attribute. The important reason for this is that we need to completely ensure that there are concepts in the folding search sub-space of an attribute. It's one important precondition of the following proposition.

Proposition 4.3 *For an ordered data context, it exists concepts in the folding search sub-space of an attribute.*

Proof : For an ordered data context $(G; M; R)$, $\forall a_i \in M$ and $a_j \in M (1 \leq j < i)$, we have $\{a_i\}' \nsubseteq \{a_j\}'$, so $\{a_i\}''$ is in the folding search sub-space of attribute a_i, otherwise $\exists a_j (1 \leq j < i), \{a_i\}' \subseteq \{a_j\}'$.

This property of ordered data context allows us to find partitions that include some search sub-space.

4.2. *A scalable algorithm: ScalingNextClosure*

We propose a new algorithm ScalingNextClosure which decomposes the search space and builds all concepts of each search sub-space. For each search sub-space, we use the same method (NextClosure algorithm) to generate the concepts. So we can generate all concepts of each search sub-space in parallel, as the search sub-spaces are independent.

ScalingNextClosure algorithm has two steps: determining the partitions (see Algorithm 1) and generating all concepts of each partition (see Algorithm 2).

For the first step of the algorithm (determining the partitions), we can decide the size of partition by a parameter DP of our algorithm according to the size of data and our needs. For the real data, we can give a value of $DP (0 < DP < 1)$. DP is used to determine the position of the beginning and the end of each partition.

We choose some attributes of ordered data context to form an order set P. If the number of the elements of P is T, we have $a_{P_1} < a_{P_2} < \ldots < a_{P_k} < \ldots < a_{P_T}$.

Algorithm 1 The first step of ScalingNextClosure algorithm: determining the partitions

input a parameter DP $(0 < DP < 1)$
generate the ordered data context (saving the order of attributes for ordered data context in an array)
output the order of attributes of the ordered data context
m = cardinal of the attribute set of the ordered data context
$min := m$
$k := 0$
while $(min >= 1)$ **do** {determining partition}
  $k++$
  $P_k := min$
  output P_k
  $min := int(min * DP)$
end while
$T := k$ //T is the number of the partitions

we denote $[a_{P_k}, a_{P_{k+1}}[$ is the search space from attribute a_{P_k} to attribute $a_{P_{k+1}}$ for ordered data context. From $\{a_{P_k}\}$ ($\{a_{P_k}\}$ is the first subset of $[a_{P_k}, a_{P_{k+1}}[$), we generate the next concepts until $\{a_{P_{k+1}}\}$, so we can find all concepts between $[a_{P_k}, a_{P_{k+1}}[$.

All concepts (non-empty) of data context are included in

$$\bigcup_{1<k<T} [a_{P_k}, a_{P_{k+1}}[\cup [a_{P_T})$$

We use all P_k to form the partitions $[a_{P_k}, a_{P_{k+1}}[$ and $[a_{P_T})$, where $1 \leq k \leq T$. Here P_k means the position of an attribute of the ordered data context, and we use it to represent the attribute a_{P_k} of the ordered data context; a_{P_k} doesn't represent the attribute of data context. When we search the concepts, Ø isn't considered.

Algorithm 2 The ScalingNextClosure algorithm to find all concepts in each partition

input the order of attributes of the ordered data context
input P_k and P_{k+1} //input the partition
$A \leftarrow \{a_{P_k}\}$
$END \leftarrow a_{P_{k+1}}$
$stop := false$
while $(!stop)$ **do**
  $A \leftarrow$ generate the next closure of A for the ordered data context
  if $END \in A$ when searching the next closure **then**
    $stop := true$
  end if
end while

For each partition, we compute the next concepts from $\{a_{P_K}\}$ to $\{a_{P_{k+1}}\}$. There is no relation between each partition. The partitions only share the same source data. We can deal with any partition independently. So we can apply this algorithm for parallel, distributed and network computing.

Here we show an example of using ScalingNextClosure algorithm to find all concepts: First, we need not to generate a data file for ordered data context, the order of attributes is only stored in the main memory. The ordered attribute set of the ordered data context for this example is: $a_1a_2a_3a_4a_5a_6a_7a_8$. And then, we give a value of the parameter to determine the partitions, for example, $DP = 0.5$. We use ScalingNextClosure algorithm to get 4 partitions: $[a_8, a_4[$, $[a_4, a_2[$, $[a_2, a_1[$ and $[a_1)$. In the end, we find all concept intents in each partition.

Ordered attributes: $a_1a_2a_3a_4a_5a_6a_7a_8$.

$DP = 0.5$ $\quad \underbrace{\underbrace{\underbrace{\underbrace{a_1}_{p_4=1}\ a_2}_{p_3=2}\ a_3\ a_4}_{p_2=4}\ a_5\ a_6\ a_7\ a_8}_{p_1=8}$

$p_1 \to a_8, p_2 \to a_4, p_3 \to a_2, p_4 \to a_1$

The partitions and their search space:

$[a_8, a_4[$: $a_8, a_7, a_7a_8, a_6, a_6a_8, a_6a_7, a_6a_7a_8, \ldots, a_5a_6a_7a_8$.

$[a_4, a_2[$: $a_4, a_4a_8, a_4a_7, a_4a_7a_8, \ldots, a_3, a_3a_8, \ldots, a_3a_4a_5a_6a_7a_8$.

$[a_2, a_1[$: $a_2, a_2a_8, a_2a_7, \ldots, a_2a_3a_4a_5a_6a_7a_8$.

$[a_1)$: $a_1, a_1a_8, a_1a_7, a_1a_7a_8, \ldots, a_1a_2a_3a_4a_5a_6a_7a_8$.

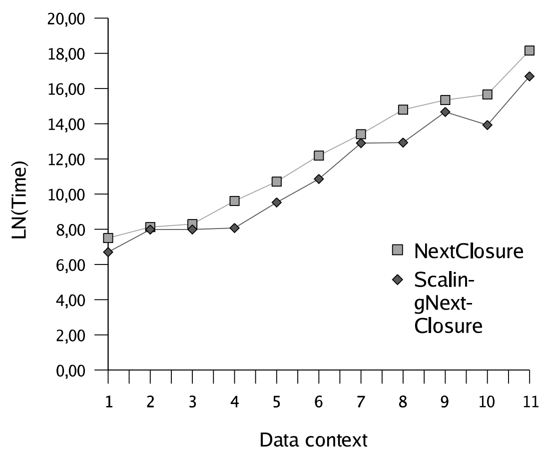
4.3. *ScalingNextClosure for worst case data*

For the worst case, a different technique can be used to generate partitions in order to avoid a great unbalance of partitions in terms of number of concepts. We can redecompose the search sub-space of an attribute into partitions so that it's easy to deal with for each partition according to the number of concepts per partition. The aim is to decrease the complexity of each partition.

5. The performance of ScalingNextClosure

We have implemented our algorithm in Java. Preliminary results of our implementation on a PIII450 computer with 128Mo RAM show that our algorithm has efficient performance. It can deal with huge data, and the total time of computing all partitions for large data is lower than that of NextClosure algorithm.

We have tested our algorithm with the datasets of the UCI repository. The comparison with NextClosure algorithm shows that our algorithm (only sequential computing) is better for large and dense data. The experimental results show that the algorithm has high performance for very large data. The total time cost of all partitions is remarkably lower than that of NextClosure algorithm. For example (see Figure 1), for the large data of UCI agaricus with 8124 objects and 124 attributes, NextClosure's time cost is 60 times higher than that of ScalingNextClosure. Some data have a large amount of attributes, and it's very hard to treat them with NextClosure, but ScalingNextClosure is very efficient in this case.



The 11 data contexts (G, M):
1: soybean-small (47,79), 2: SPECT (187,23), 3: flare2 (1066,32), 4: audiology(26,110), 5: breast (699,110), 6: lung-cancer (32,228), 7: promoters (106,228), 8: soybean-large (307,133), 9: dermatology (366,130), 10: nursery (12960,31), 11: agaricus (8124,124)

Figure 7. *Performance of comparison for Next Closure and ScalingNextClosure algorithms on UCI data. The time cost (in milliseconds) is represented by LN(Time).*

For our experiments, we have used ScalingNextClosure algorithm to generate various different partitions for some datasets. For example, we can build different partitions according to the size of data. Figure 2 shows the results of comparisons with different values of parameter DP. Given a bigger value, we can build more partitions. Varying different values of DP can affect the result of our algorithm, as it is the case with Agaricus dataset. How many partitions and what partition should we create for the best performance? We will try to find an answer to this question in our future work.

We have tested our algorithm in the worst case, and it succeeds in computing some large data that were impossible to be computed with other algorithms. For exam-

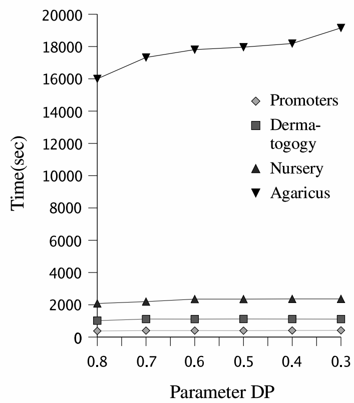


Figure 8. *Performance comparison on different values of parameter DP with 4 datasets.*

ple: the worst case with 30 attributes is very hard to compute with other algorithms [FU 03a].

Using ScalingNextClosure algorithm, we have generated all concepts for worst case data sets with 24, 25, 30, 35 and 50 attributes.

6. Conclusion

The concept lattice algorithm to generate concepts or diagram graph is considered important in theory and for its application. We need algorithms of high level performance to satisfy the mining and learning task. Four algorithms are analyzed and are compared in this paper, of course, this work will be extended to other lattice algorithms. We use real dataset and worst cases datasets to test four algorithms in Java environment, the analysis shows that algorithms of Ganter and then Bordat are faster than others. Ganter's algorithm is the best for large and dense data. Bordat's algorithm can be used to generate the line diagram if the computer has enough memory.

In this paper we discuss for the first time dual algorithm for concept lattices. The difference between algorithm and its dual algorithm is presented. We should consider duality when comparing lattice algorithms.

Even if this work shows performance of concept lattices algorithm in ML benchmarks, and Ganter's algorithm is the best for large and dense data, the existing lattice algorithms are still difficult to deal with large data. In fact, The search space of concepts is very large for large data. It's a hard problem to determine all the concepts of large data. We need to develop faster algorithm, or to improve existing algorithms, to raise the efficiency of concept lattice for data mining [FU 03b, VAL 02].

So furthermore we study the search space of concepts in order to partition the search space to scale up lattice algorithm. A new efficient scalable lattice-based algorithm, ScalingNextClosure is proposed for creating the partitions of the search space and building concepts in each partition. ScalingNextClosure is different from other existing decomposition algorithms that generate concept lattice using the approach of context decomposition [VAL 02], which is based on an incremental approach.

The experimental results show that ScalingNextClosure algorithm is very suitable and scalable to deal with large data. For the ongoing research, we will parallelize ScalingNextClosure in order to improve its performance. Futhermore, we will extend our method to classification and association rules mining.

7. References

[BAR 70] BARBUT M., MONJARDET B., *Ordre et classification — Algèbre et combinatoire (2 tomes)*, Hachette, 1970.

[BIR 67] BIRKHOFF G., *Lattice Theory*, American Mathematical Society, Providence, RI, 3rd edition, 1967.

[BLA 98] BLAKE C., KEOGH E., MERZ C., "UCI Repository of machine learning databases", 1998, http://www.ics.uci.edu/∼mlearn/MLRepository.html.

[BOR 86] BORDAT J., "Calcul pratique du treillis de galois d'une correspondance", *Mathématiques, Informatiques et Sciences Humaines*, vol. 24, num. 94, 1986, p. 31-47.

[CAR 96] CARPINETO C., ROMANO G., "A Lattice Conceptual Clustering System and its Application to Browsing Retrieval", *Machine Learning*, vol. 24, 1996, p. 95-122.

[CHE 69] CHEIN M., "Algorithme de recherche des sous-matrice premières d'une matrice", *Bulletin Math. de la Soc. Sci. de la R.S. de Roumanie*, vol. 61, num. 1, 1969, Tome 13.

[FU 03a] FU H., MEPHU NGUIFO E., "How well go Lattice Algorithms on currently used Machine Learning TestBeds?", *ICFCA 2003, First International Conference on Formal Concept Analysis*, 2003.

[FU 03b] FU H., MEPHU NGUIFO E., "Partitioning large data to scale up lattice-based algorithm", *Proceedings of ICTAI03*, Sacramento, CA, November 2003, IEEE Computer Press.

[FU 04] FU H., MEPHU NGUIFO E., "A parallel algorithm to generate formal concepts for large data", *ICFCA 2004, Second International Conference on Formal Concept Analysis*, 2004.

[GAN 84] GANTER B., "Two basic algorithms in Concept Analysis", report num. 831, 1984, Technische Hochschule, Darmstadt, Germany, preprint.

[GAN 99] GANTER B., WILLE R., *Formal Concept Analysis. Mathematical Foundations*, Springer, 1999.

[GOD 95] GODIN R., MINEAU G., MISSAOUI R., MILI H., "Méthodes de classifi cation conceptuelle basées sur les treillis de Galois et applications", *Revue d'intelligence artificielle*, vol. 9, num. 2, 1995, p. 105-137.

[GOD 98] GODIN R., CHAU T.-T., "Comparaison d'algorithmes de construction de hiérarchies de classes", report , 1998, Université de Québec.

[GU9̃0] GUÉNOCHE A., "Construction du treillis de Galois d'une relation binaire", *Mathématiques et sciences humaines*, vol. 109, 1990.

[KUZ 02] KUZNETSOV S., OBIEDKOV S., "Comparing Performance of Algorithms for Generating Concept Lattices", *JETAI Special Issue on Concept Lattice for KDD*, vol. 14, num. 2/3, 2002, p. 189-216, Talor & Francis Group.

[Mep 02] MEPHU NGUIFO E., LIQUIERE M., DUQUENNE V., *JETAI Special Issue on Concept Lattice for KDD*, Taylor and Francis, 2002.

[NOR 78] NORRIS E., "An algorithm for computing the maximal rectangles in a binary relation", *Revue Roumaine Math. Pures et Appl.*, vol. XXIII, num. 2, 1978, p. 243-250.

[NOU 99] NOURINE L., RAYNAUD O., "A Fast Algorithm for Building Lattices", *Information Processing Letters*, vol. 71, 1999, p. 199-204.

[RIO 03] RIOULT F., BOULICAUT J.-F., CRÉMILLEUX B., BESSON J., "Using transposition for pattern discovery from microarray data", *Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery*, ACM Press, 2003, p. 73–79.

[VAL 02] VALTCHEV P., MISSAOUI R., LEBRUN P., "A partition-based approach towards constructing Galois (concept) lattices", *Discrete Mathematics*, vol. 256, num. 3, 2002, p. 801-829.