A Parallel Algorithm to Generate Formal Concepts for Large Data

Huaiguo Fu and 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. One of the most effective methods to deal with large data for data analysis and data mining is to develop parallel algorithm. Although Formal concept analysis is an effective tool for data analysis and knowledge discovery, it's very hard for concept lattice structures to face the complexity of very large data. So we propose a new parallel algorithm based on the NextClosure algorithm to generate formal concepts for large data.

1 Introduction

Parallel algorithms and parallel computer provide a very effective method to deal with very large data for data analysis and data mining [11]. The design of parallel algorithm for very large data analysis and processing becomes the necessity of many enterprises. In fact, many sequential algorithms can be improved by parallel algorithms [8].

Concept lattice is considered as an effective tool for data analysis, data mining, machine learning, information retrieval, etc. The generation of concepts and concept lattices is an essential task for formal concept analysis (FCA). Several algorithms were proposed to generate concepts or concept lattices of a context, for example: Bordat, Ganter (NextClosure), Chein, Norris, Godin and Nourine, etc [6]. Experimental comparisons of these algorithms show that NextClosure algorithm [4] is one of the best for large and dense data [6, 2]. But the problem is that it still takes very high time cost to deal with huge data. One way to avoid this problem is to design parallel lattice-based algorithms. In this paper, we propose a new parallel lattice-based algorithm **ParallelNextClosure**.

ParallelNextClosure is based on the decomposition of search space to generate concept. It's different from other existing decomposition algorithms that generate concept lattice using the approach of context decomposition [9] which could build many overlapping search sub-spaces. ParallelNextClosure uses a new method to decompose the search space by generating different non-overlapping search subspaces. It is also different from ParGal algorithm[7]. ParGal is based on Bordat's algorithm, there are many communications within the processors. Our algorithm is based on NextClosure algorithm. It can freely decompose the search space in different partitions if there exists at least one concept in each partition. Each

P. Eklund (Ed.): ICFCA 2004, LNAI 2961, pp. 394–401, 2004.

partition is independent and only shares the same source data (context) with other partitions. We can generate concepts independently for each partition using one processor. ParallelNextClosure algorithm shows good scalability, and can be easily used for parallel and distributed computing.

We consider the reader familiar with the basic notions of FCA such as context, Galois connection, closure operator, concept and concept lattice [1, 10, 5].

The rest of this paper is organized as follows: A new parallel algorithm ParallelNextClosure will be proposed in the next section. The algorithm analysis will be discussed in section 3. The paper ends with a short conclusion in the last section.

2 Parallel Lattice-Based Algorithm

2.1 NextClosure Algorithm

The principle of the NextClosure algorithm uses the characteristic vector, which represents arbitrary subsets A of M, to enumerate all concepts for context (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 set is $\emptyset''$. The NextClosure algorithm proved that if we know an arbitrary set A, the next concept[1] 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.

The NextClosure algorithm can generate concepts more rapidly than other lattice algorithms for large data, but it still takes too much time to deal with very large data. So we improve it and propose a new algorithm to deal with very large data, our main idea is to use a parallel method to generate concepts.

2.2 ParallelNextClosure Algorithm

We propose a new algorithm which divides the search space into certain partitions. Each partition contains at least one concept. The processing of each partition is independent from another one. This is the main idea of our algorithm ParallelNextClosure in order to reduce the communication cost.

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

[1] It is the smallest one of all concepts that is larger than A.

Definition 1. *A context* (G, M, R) *is called* **ordered context**, *if the attribute set* $M = \{a_1, \ldots, a_i, \ldots, a_{m-1}, a_m\}$, *the attributes with the same objects are merged as one attribute, and for all* $i, j <= m$ *: if the size of the extent of* a_i *is smaller than the size of the extent of* a_j, *then* $i < j$.

In the cross table of such an ordered context we take the first column for an attribute with minimal extent size and the last column for an attribute with maximal extent size. An ordered context has the same concept lattice or concept lattice as the context, it doesn't change any characteristic of the context. The ordered context is very easy to be generated from a context. For Parallel-NextClosure algorithm, we don't generate a data file for ordered context, the ordered context is only stored in main memory. We can prove that this arrangement can raise efficiency of our algorithm.

Partitioning the Search Space for Concepts. In fact, a concept is a subset of M, so all subsets of M are elements of the search space. So the size of search space for enumeration of all concepts is 2^m. This search space can be considered as the fold of some subsets of M. For example, we consider that the search space is formed by $folds$, where $fold_i$ is all subsets of $\{a_i, \ldots, a_{m-3}, a_{m-2}, a_{m-1}, a_m\}$ that include a_i. Each $fold$ is a search sub-space. Here we define **folding search sub-space** in order to decompose the search space.

Definition 2. *Let the attribute set* M *of a context* (G, M, R) *be* $\{a_1, \ldots, a_i, \ldots, a_{m-1}, a_m\}$. *Given* $\forall a_i \in M$, *let* $S(a_i) = \{$ *all subsets of the set* $\{a_i, a_{i+1}, \ldots, a_{m-1}, a_m\}$ *which contain* a_i $\}$, $S(a_i)$ *is called* **folding search sub-space (F3S) of** a_i.

For example, the folding search sub-space of attribute a_{m-3} is $\{a_{m-3}\}$, $\{a_{m-3}, a_m\}$, $\{a_{m-3}, a_{m-1}\}$, $\{a_{m-3}, a_{m-1}, a_m\}$, $\{a_{m-3}, a_{m-2}\}$, …, $\{a_{m-3}, a_{m-2}, a_{m-1}, a_m\}$, i.e. all subsets of $\{a_{m-3}, a_{m-2}, a_{m-1}, a_m\}$ that include a_{m-3}.

Proposition 1. *Let* (G, M, R) *be a context,* $M = \{a_1, \ldots, a_i, \ldots, a_{m-1}, a_m\}$, $\forall a_i \in M$, $n =$ *the number of objects of attribute* a_i, *then the size of the folding search sub-space (for concept) of* a_i *is the minimum of* 2^n *and* 2^{m-i}.

Proof: $\forall a_i \in M$, *the set of* $\{a_{i+1}, a_{i+2}, \ldots, a_{m-2}, a_{m-1}, a_m\}$ *has* $m - i$ *attributes. So according to the definition, the size of F3S of* a_i *is* 2^{m-i}.

On the other hand, if the number of objects that verify attribute a_i *is n, we have* 2^n *object subsets corresponding to* a_i. *So there are at most* 2^n *concepts that include attribute* a_i.

Thus the size of the folding search sub-space of a_i *is the minimum of* 2^n *and* 2^{m-i}.

So we can order the context with the number of objects of each attribute. The smallest attribute is in the first column, and the biggest one is in the last column. In practice, this arrangement can remarkably allow to build an efficient algorithm for real data.

In the definition of ordered context, we need to merge the attributes with exactly the same objects as one item. 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 2. *For an ordered context, there are concepts in the folding search sub-space of an attribute.*

Proof: For an ordered context (G, M, R), $\forall a_i \in M$ *and* $a_j \in M(1 \leq j < i)$, *we have* $\{a_i\}' \not\subseteq \{a_j\}'$, *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 context allows us to find partitions that include some search sub-space.

A Parallel Algorithm. We choose some attributes of ordered context to form an order set P. If the number of the elements of P is T, we have $a_{P_1} < a_{P_2} < \ldots < a_{P_k} < \ldots < a_{P_T}$. We denote $[a_{P_k}, a_{P_{k+1}}]$ as all F3S of attributes from a_{P_k} to $a_{P_{k+1}}$ for ordered 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 that we can find all concepts between $[a_{P_k}, a_{P_{k+1}}]$.

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

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

We propose a new algorithm ParallelNextClosure 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 closed sets. So we can generate all concepts of each search sub-space in parallel, as the search sub-spaces are independent.

ParallelNextClosure algorithm has two steps : determining the partitions and generating all concepts in each partition

In the first step of the algorithm, we can decide the number of partitions by a parameter $DP(0 < DP < 1)$ according to the size of data and our need. DP is used to determine the position of the beginning and the end of each partition.

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 context, and we use it to represent the attribute a_{P_k} of the ordered context. a_{P_k} has a corresponding position in the initial context. When we search for the concepts, $\emptyset$ isn't considered.

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.

Proposition 3. *Each partition is independent. In other words, we can find all concepts of one partition independently.*

Algorithm 1 Determining the partitions by the first processor

```
1: input a parameter DP (0 < DP < 1)
2: generate the ordered context (saving the order of attributes for ordered context in
   an array)
3: output the order of attributes of the ordered context
4: m = cardinal of the attribute set of the ordered context
5: min := m
6: k := 0
7: while (min >= 1) do {determining partition}
8:    k := k + 1
9:    P_k := min
10:   output P_k
11:   min := int(min * DP)
12: end while
13: T := k      //T is the number of the partitions
```

Algorithm 2 Generating all concepts in each partition for one processor

```
1: input the order of attribute of ordered context
2: input P_k and P_{k+1}      //input the partition
3: A ← {a_{P_k}}
4: END ← a_{P_{k+1}}
5: STOP := false
6: while (!STOP) do
7:    A ← generate the next closure of A for the ordered context
8:    if END ∈ A when searching the next closure then
9:       STOP := true
10:   end if
11: end while
```

Proof: When we find the concepts of a partition, we only need to know the first subset and the last subset of this partition. And then we build next concepts from the first subset until the last subset, the computing is closed in this partition. So each partition is independent.

Proposition 4. *All concepts of a context are included in the partitions. In other words, ParallelNextClosure algorithm generates the same concept lattice as NextClosure algorithm.*

Proof: The partitions are obtained by partitioning the search space of all concepts. So all concepts of a context are included in the partitions.

Proposition 5. *ParallelNextClosure algorithm always ends.*

Proof: Each partition has a last subset, so it can end when generating all concepts in each partition with one processor. So ParallelNextClosure algorithm always ends.

We show below an example (using Figure 1) of ParallelNextClosure algorithm:

First, we need to generate the ordered context, the attribute set of the ordered context is: $a_5a_8a_6a_7a_4a_3a_2a_1$(a_2 and a_9 are merged as a_2). And then, we can give a value to the parameter to determine the partitions, for example, $DP = 0.5$. We get 4 partitions: $[a_1, a_7]$, $[a_7, a_8]$, $[a_8, a_5]$ and $[a_5)$. At the end, we find all concepts in each partition. In fact, the search space of each partition is:

$[a_1, a_7]$: a_1, a_2, a_2a_1, a_3, a_3a_1, a_3a_2, $a_3a_2a_1$, a_4, a_4a_1, ..., $a_4a_3a_2a_1$

$[a_7, a_8]$: a_7a_1, a_7a_2, $a_7a_2a_1$, ..., a_6, a_6a_1, ..., $a_6a_7a_4a_3a_2a_1$

$[a_8, a_5]$: a_8, a_8a_1, a_8a_2, $a_8a_2a_1$, ..., $a_8a_6a_7a_4a_3a_2a_1$

$[a_5)$: a_5, a_5a_1, a_5a_2, ..., $a_5a_8a_6a_7a_4a_3a_2a_1$

3 Algorithm Analysis

In fact, ParallelNextClosure algorithm is suitable for real data. The worst case should be considered when we determine the partitions : the worst case appears when the sizes of G and M are equal to m, and each attribute is verified by $m-1$ different objects, each object possesses $m-1$ different attributes.

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 the number of concepts. We can redecompose the search sub-space of an attribute into some 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.

The problem of balance is always an important problem for parallel algorithm. The parameter $DP(0 < DP < 1)$ of ParallelNextClosure algorithm is used just to balance the partition size. The computing of each partition can be implemented on any free processor so that it can help to balance the processors.

4 Conclusion

In this paper, a new parallel lattice-based algorithm, ParallelNextClosure is proposed for creating the partitions and building concepts in each partition.

ParallelNextClosure uses a new method to decompose the search space. We have defined ordered context and folding search sub-space of an attribute. Furthermore, we have proved that there are concepts in the folding search sub-space of an attribute. Thus our method can freely decompose the search space in any partitions if there are concepts in each partition. Each partition is independent, so we can realize our parallel algorithm.

Experimentations with one processor show the following results [3]: on some data, our algorithm was 60 times faster than NextClosure. And we have generated all concepts for worst case data sets with 24, 25, 30, 35 and 50 attributes, which was very difficult to obtain with NextClosure on a 128MB RAM computer. The gain comes from lack of memory in the case of NextClosure.

The ongoing research is to study the problem of balance of all partitions. Further research will consist of implementation of this algorithm on a parallel platform and an analysis of the results.

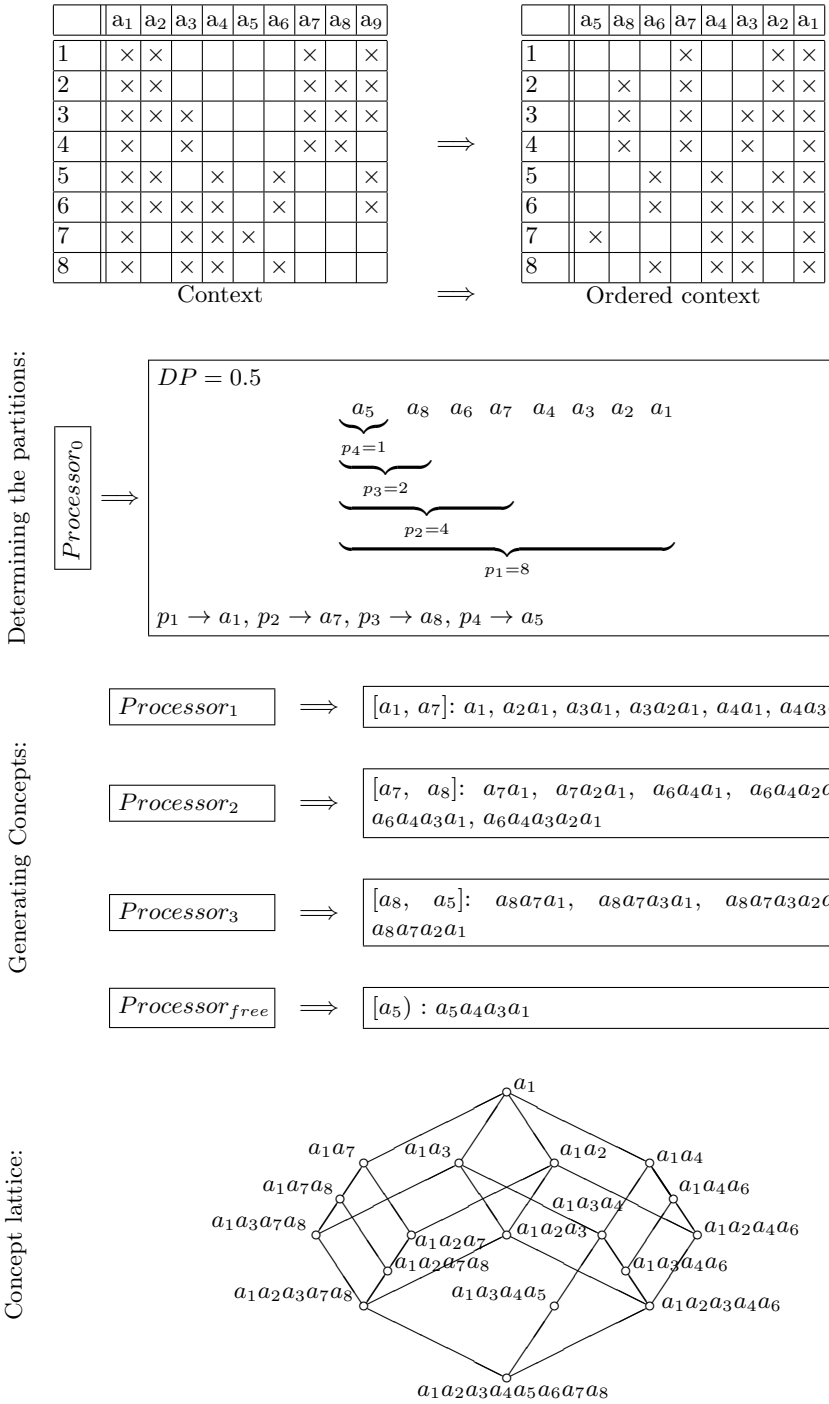

Fig. 1. An example with ParallelNextClosure algorithm using 3 processors

Acknowledgements

We are grateful to anonymous reviewers for helpful comments and to Jean Jacques Givry for english proofreading. This research benefits from the support of IUT de Lens, Université d'Artois, CNRS and the region Nord/Pas de calais.

References

1. G. Birkhoff. *Lattice Theory.* American Mathematical Society, Providence, RI, 3rd edition, 1967.
2. Huaiguo Fu and Engelbert Mephu Nguifo. How well go lattice algorithms on currently used machine learning testbeds? In *ICFCA 2003, First International Conference on Formal Concept Analysis*, 2003.
3. Huaiguo Fu and Engelbert Mephu Nguifo. Partitioning large data to scale up lattice-based algorithm. In *Proceedings of ICTAI03*, Sacramento, CA, November 2003. IEEE Computer Press.
4. B. Ganter. Two basic algorithms in concept analysis. Technical Report 831, Technische Hochschule, Darmstadt, Germany, 1984. preprint.
5. B. Ganter and R. Wille. *Formal Concept Analysis. Mathematical Foundations.* Springer, 1999.
6. S. Kuznetsov and S. Obiedkov. Comparing performance of algorithms for generating concept lattices. *JETAI Special Issue on Concept Lattice for KDD*, 14(2/3):189–216, 2002.
7. P. Njiwoua and E. Mephu Nguifo. A parallel algorithm to build Concept Lattice. In *Proceedings of the* 4^{th} *Groningen International Information Technology Conference for Students*, pages 103–107, University of Groningen, The Netherlands, Février 1997.
8. M. Tchuente. *Parallel Computation on Regular Arrays.* Algorithms and Architectures for Advanced Scientific Computing. Manchester University Press, 1991.
9. Petko Valtchev and Rokia Missaoui. Building galois (concept) lattices from parts: Generalizing the incremental approach. In *Proceedings of the ICCS 2001, LNCS 2120*, pages 290–303. Springer Verlag, 2001.
10. R. Wille. Restructuring Lattice Theory. In Ivan Rival, editor, *Symposium on Ordered Sets*, pages 445–470. University of Calgary, Boston, 1982.
11. M.J. Zaki and C.-T. Ho. *Large-Scale Parallel Data Mining.* Springer, 2000.