Parallel Computation of Reducts

Robert Susmaga

Institute of Computing Science, Poznań University of Technology,
Piotrowo 3A, 60-965 Poznań, Poland.
Robert.Susmaga@CS.PUT.Poznan.PL

Abstract. The paper addresses the problem of reduct generation, one of the key issues in the rough set theory. A considerable speed up of computations may be achieved by decomposing the original task into subtasks and executing these as parallel processes. This paper presents an effective method of such a decomposition. The presented algorithm is an adaptation of the reduct generation algorithm based on the notion of discernibility matrix. The practical behaviour of the parallel algorithm is illustrated with a computational experiment conducted for a real-life data set.

1 Introduction

This paper addresses the problem of reduct computation in information systems. The reduct is a notion that has been given much attention in numerous papers within the Rough Set community [4,5,6,7,9]. The idea of reduct and attribute reduction in information tables is, in general, related to a broader problem of feature selection, which has been the focus of many papers in the area of Machine Learning (see [1] for a comprehensive review of feature selection methods). A successful application of rough set reducts has been reported e.g. in [3].

One of the most challenging issues related to reducts is the problem of reduct generation. This is because generating reducts is a computationally complex task. The problem of generating a minimal reduct has been proved to be NP-hard in [7]. This result directs the line of research towards different methods of handling the problem of computational complexity. As a result, the reduct generating procedures employ both exact and heuristic methods.

This paper deals with exact methods. The forthcoming sections provide motivation for parallelization of reduct generation and introduce a parallel algorithm for the computation of all exact reducts. The algorithm is based on the decomposition of an exact algorithm from [7]. Because introducing the framework for formal definition of reducts is impossible due to the paper size restrictions, the reader is referred to [4,5,7,9].

The presented methodology is certainly not the only existing approach to the problem of decomposition in reduct generation. It focuses on a single data set and the parallelization is introduced mainly to speed up the computations for this data set. An earlier approach from [4] considers information systems in which the data are distributed among several locations. The process of reduct

L. Polkowski and A. Skowron (Eds.): RSCTC'98, LNAI 1424, pp. 450–458, 1998.

computation may then proceed independently in all of those locations and the results may be combined together to produce the final set of reducts.

The rest of the paper is organized as follows. Section 2 presents a sequential reduct generation algorithm. Section 3 introduces the parallelization of this algorithm. In Section 5 results of the experiment aimed at providing practical evaluation of the parallel algorithm are presented and discussed. The last section draws attention to some unsolved problems and outlines directions of the future research on the subject.

2 The Sequential Implementation of the Algorithm

The original algorithm from [7] is based on the notion of the discernibility matrix. The result of the algorithm is the set of all exact reducts. The main notion of the algorithm, the discernibility matrix, is defined as follows. Given the information table $IT =< U, Q, V, d >$, $|U| = N$, the discernibility matrix is defined as the matrix $[C_{i,j}]$, $i = 1..N$, $j = 1..N$, where:
$C_{i,j} := \{q \in Q : d(u_i, q) \neq d(u_j, q),\ for\ each\ pair\ i, j\}$.

Input: A set of objects U ($|U|=N$) described by values of attributes from the set Q.

Output: The set K of all (absolute) reducts for the set U.

PHASE I
Step 1
Create the absorbed discernibility list ADL for $i=1..N$, $j=1..N$, eliminating empty and non-minimal elements:
$ADL:=\{C_{ij}: C_{ij}\neq\emptyset$ and for no $C_{lm}\in ADL: C_{lm}\subset C_{ij}\}$, where:
$C_{ij}:=\{q\in Q: \delta(u_i,q)\neq\delta(u_j,q)$, for each pair $i,j\}$.
The resulting absorbed discernibility list contains elements $(C_1, C_2, \ldots, C_d)$, where $d\in[1, N(N-1)/2]$.

Step 2
Sort the ADL in the ascending order of the cardinality of its elements.

PHASE II
Step 1
$R_0:=\{\emptyset\}$.

Step 2
For $i=1..d$ compute:

$S_i:=\{R\in R_{i-1} : R\cap C_i\neq\emptyset\}$.

$T_i:=\bigcup_{q\in C_i}\ \bigcup_{R\in R_{i-1}: R\cap C_i\neq\emptyset}\{R\cup\{q\}\}$.

$MIN_i:=\{R\in T_i: Min(R,ADL,i)=true\}$.

$R_i:=S_i\cup MIN_i$.

The final result is $K:=R_d$.

Fig. 1. The Modified Reduct Generation Algorithm (MRGA)

The main idea of the method is to find all minimal attribute subsets that have non-empty intersections with all (non-empty) elements of the discernibility matrix. The algorithm consists of two phases: the first phase creates the elements of the discernibility matrix, and the second one generates reducts using these

elements. To improve efficiency, the elements of the discernibility matrix are stored in form of an absorbed lists. In the last step of the first phase, what is a modification of the original algorithm, the absorbed list is sorted in the ascending order of the cardinality of its elements. The modified reduct generating algorithm (MRGA) is presented in Figure 1.

In the algorithm, Min performs an operation that corresponds to checking for prime implicants of a Boolean function. The returned value is *true* if the argument R does not contain redundant attributes. $ADL_{1..i}$ is a list of i initial elements of ADL: $ADL_{1..i} = (C1, C2, ..., Ci)$. Thanks to the links established in [7], the algorithm may be also directly applied to the problem of searching for all prime implicants of Boolean functions.

In the form given above the algorithm searches for so called absolute reducts. This is determined by the definition of the discernibility matrix, which is computed with a uniform treatment of all attributes from Q. If the split of Q into condition (C) and decision (D) attributes is to be taken into account and so called relative reducts are to be produced, the discernibility matrix should be computed as follows:
$C_{i,j} := \{q \in Q : d(u_i, q) \neq d(u_j, q)\ and\ (u_i, u_j) \notin IND(D),\ for\ each\ pair\ i, j\}$, where $(u_i, u_j) \notin IND(D)$ means that the objects u_i and u_j belong to different classes defined by the decision attributes.

3 The Parallel Implementation of the Algorithm

The most general idea of parallelization is decomposing the computing task into several subtasks which may be then processed at the same time on different processors. This requires, first of all, a specialized algorithm. The main objective of parallelization is reducing the computing time. Another important benefit may be the reduction of memory requirements (see [2] for a detailed description of different aspects of parallel algorithms).
Parallelization of MRGA is based on its following characteristics.

Observation In each iteration of the *Step 2* (*PHASE II*) of the algorithm the following holds: $S_i \cap T_i = \emptyset$; every $R \in S_i$ is a copy of an $R' \in R_{i-1}$; every $R \in T_i$ is created from a single $R' \in R_{i-1}$ by augmenting it by an element $q \in Q$; MIN_i is a subset of T_i created by discarding some of its elements — the process is conducted on the basis of the index i and the value of $ADL_{1..i}$.

Conclusion 1 Any two $R, R' \in R_i$ are independent in the sense that no information on R or on how it has been created is required for producing R'.
Conclusion 2 The process of subset creation has a tree-like structure: in each iteration an element $R \in R_i$ may be either copied to the next iteration without changes or be deleted, giving rise to a collection of its proper supersets, which are subsequently processed independently of one another and of the other elements of R_i.

Input: A set of objects U ($|U|=N$) described by values of attributes from the set Q;
the subtask count SC, the branching factor BF (it is assumed that $BF \geq SC$).

Output: The set K of all (absolute) reducts for the set U.

PHASE I – As in Modified Reduct Generating Algorithm

PHASE II
Step 1
$R_0 := \{\emptyset\}$.
$i := 1$.

Step 2
While ($|R_0| < BF$) *and* ($i \leq d$) do loop :
$S_i := \{R \in R_{i-1} : R \cap C_i \neq \emptyset\}$.
$T_i := \bigcup_{q \in C_i} \bigcup_{R \in R_{i-1}: R \cap C_i \neq \emptyset} \{R \cup \{q\}\}$.
$MIN_i := \{R \in T_i : Min(R, ADL, i) = true\}$.
$R_i := S_i \cup MIN_i$.
$i := i+1$.

Step 3
If ($i > d$) then set $K := R_d$ and stop the algorithm.
Otherwise proceed with *Step 4*.

Step 4
Partition the set R_i into SC subsets R_i^j, ($j=1..SC$), such that $R_i^k \cap R_i^l = \emptyset$ for each $k \neq l$,
$\bigcup_{j=1}^{SC} R_i^j = R_i$ and the subsets R_i^j are of approximately equal size.

Step 5
Execute SC parallel subtasks, each performing the procedure $K^j := SUB(i, d, R_i^j, ADL)$, for $j=1..SC$.
Set $K := \bigcup_{j=1}^{SC} K^j$ and stop the algorithm.

Procedure *SUB* (*InitialIndex* , *FinalIndex* , *PartialSolution* , *AbsorbedDiscernibilityList*)

Step 1
$R_{InitialIndex-1} := PartialSolution$.

Step 2
For $i = InitialIndex..FinalIndex$ compute:
$S_i := \{R \in R_{i-1} : R \cap C_i \neq \emptyset\}$.
$T_i := \bigcup_{q \in C_i} \bigcup_{R \in R_{i-1}: R \cap C_i \neq \emptyset} \{R \cup \{q\}\}$.
$MIN_i := \{R \in T_i : Min(R, ADL, i) = true\}$.
$R_i := S_i \cup MIN_i$.

Return $R_{FinalIndex}$ as the result of the procedure.

Fig. 2. The Parallel Implementation of MRGA

The conclusions lead to a formulation of the parallel algorithm for reduct computation. After completing the *PHASE I* of the sequential algorithm and performing several iterations of the *Step 2* (*PHASE II*), the resulting R_i may be split into a family of subsets which create a partition of the set R_i, i.e. all of them are pairwise disjoint and they all sum up to the set R_i. From now on the computations may continue in form of several subtasks, each of which proceeds with exactly one partition substituted for the whole set R_i. It is important that the resulting subtasks are absolutely independent of one another. The final set of reducts is the sum of the sets of reducts generated by each subtask.

The implementation of the parallel algorithm requires two additional parameters: the number of subtasks which are to be initiated (subtask count, SC) and the cardinality of the set R_i which is to be reached before the algorithm branches into multiple subtasks (branching factor, BF). The branching factor must be equal to or greater than the number of subtasks so that each subtask may start with at least one element of R_i. The role of this parameter is closely discussed in Section 5.

There are no formal restrictions as to the method of partitioning the set R_i into disjoint subsets. In the described experiments this was implemented simply as assigning the attribute subset number n to the subtask number $m = (n \ mod \ SC) + 1$.

4 Experimental Evaluation of the Parallel Implementation

The data set used in the experiment was a medical file containing descriptions of 343 patients after the ESWL treatment [8]. Each patient is described by a vector of 33 condition attributes and one decision attribute (defining two decision classes). The file was selected for presenting the results of the experiments only because of its computational characteristics: a large number of relative reducts (38207), a moderate computing time (327 seconds) and small memory requirements (about 380 kB). It is stressed that this paper contains no claims as to any potential suitability of the selected data file for reduct generation or the usefulness of the generated reducts in further analyses.

The practical behaviour of the parallel implementation may be best characterized by plotting the computing times (understood as the maximal computing time of all of the subtasks + the time required to reach the branching point) against the increasing number of subtasks employed (see Figure 3). The same applies to the maximal memory requirements (Figure 4).

The main computing platform in the experiments was a SUN Workstation equipped with a Sparc4 processor running at 110 MHz.

Computing Times

Inspection of Figure 3 allows to conclude that the parallel implementation of the algorithm reduces the computing time considerably. The fact that the decrease of the computing time is worse than linear may be explained as follows.

The problem of decomposing the original problem into subtasks is unbalanced in the sense that the number of reducts generated by each of the subtasks is highly variable. It results from the fact that it is impossible to determine in advance how many reducts will be finally generated by a given subtask, what, in turn, results from the highly variable 'productivity' of different attribute subsets initially assigned to the subtask. Experiments indicate that running the algorithm with the value of the branching factor equal to the number of subtasks (which is the lowest possible value of this parameter) is highly unbalanced, including situations where the maximal subtask computing times are close to the computing time of the sequential implementation. A solution to this seems to be increasing the value of the branching factor. This results, first of all, in a direct increase of the initial computing time (it takes longer to reach the branching point of the algorithm) but it has also a positive effect on balancing the subtasks and, in longer perspective, proves to be beneficial.

The general conclusion is that the computing time decreases satisfactorily only after the branching factor is set to a value which enables each of the subtasks to start with at least several attribute subsets. This is, however, only an improvised, temporary principle — discovering optimal (provided there exists one) or close to optimal value of the branching factor needs further research.

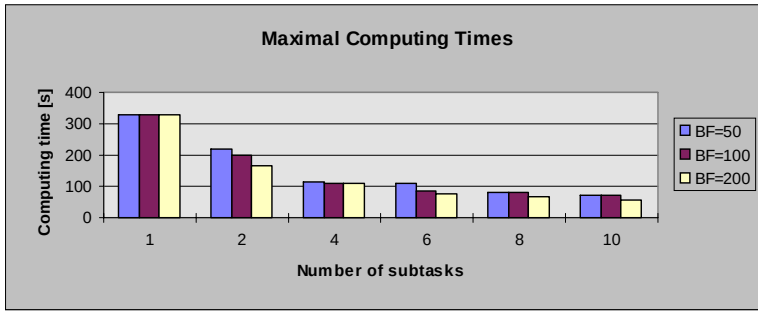

Fig. 3. Maximal computing times against the increasing number of subtasks and different branching factors

Memory Requirements

Another very important feature of the parallel implementation is the characteristics of its memory requirements. Inspection of Figure 4 reveals that the maximal memory requirements of the sequential algorithm are considerably reduced by its decomposition into subtasks, the reduction being most evident for a small number of subtasks and becoming less evident with more subtasks. The

fact that the decrease of memory requirements is worse than linear may be explained as follows. There are two main memory-consuming structures of the algorithm: the set R_i and the initial (unabsorbed) form of $ADL_{1..i}$. Usually the size of R_i exceeds considerably that of the unabsorbed list, but after partitioning of R_i the size of the unabsorbed list starts to dominate, obliterating the effects of partitioning.

Applying parallelization with small numbers of subtasks, however, still seems to be a very good alternative, especially in computing environments where the memory proves to be the bottle-neck of the process.

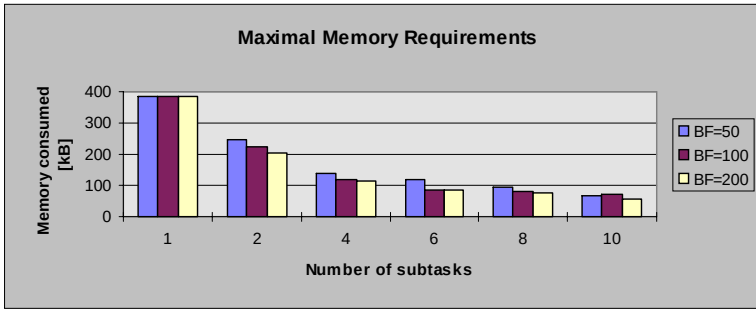

Fig. 4. Maximal memory requirements against the increasing number of subtasks and different branching factors

5 Conclusions and Directions of Future Research

This paper introduces a parallel algorithm for computing all exact reducts of a decision table. The algorithm is based on an algorithm described originally in [7]. Results of experiments aimed at verifying the algorithm's practical behaviour are also presented and discussed. The results indicate usefulness of the parallel implementation, which allows for the reduction of both the computing time and memory requirements of the reduct generating process.

There still remain some issues which need further research. The main of them are: establishing the number of subtasks to be started and balancing the subtasks so that they generate approximately equal numbers of reducts and exhibit similar computing times and memory requirements.

In its current version, the parallel algorithms branches at one point into an arbitrary number of subtasks. Such a solution does not take into account the computational needs of the particular data set for which the algorithm has been invoked, because it is not possible, in general, to determine the final number of reducts in advance. In an alternative approach, the number of subtasks would

not have to be specified — the only control parameter being the branching factor. Upon reaching it, the algorithm would branch into two parallel subtasks, with each of them proceeding with a half of the current load. The same branching procedure would be subsequently applied recursively in both resulting subtasks.

A related problem is that of balancing the resulting subtasks. Presently, the parallel algorithm assigns the initially created attribute subsets one by one to consecutive subtasks. Because the original ordering of the attribute subsets is random, this partitioning may also be viewed as random. It must be stressed, however, that the ordering of the absorbed discernibility list, from which the attribute subsets are created, is not random — the list is ordered in ascending order of its elements. As a result, the initial attribute subsets contain those attributes that occurred in low-cardinality elements of the discernibility list. A better balancing scheme could incorporate some attribute counts, so that the numbers of different attributes in the initial subsets would be approximately the same.

Acknowledgements

This paper has been supported by the grant KBN no 8T11C–013–13. Parts of the reported computations were conducted at the Supercomputing and Networking Centre of Poznań, Poland. Special thanks are due to Darek Wawrzyniak from the Institute of Computing Science for his eager help on parallelization issues.

References

1. Dash M. & Liu H. 'Feature Selection for Classification', *Intelligent Data Analysis* (on-line journal), Vol. 1, No 3, (1997), `http://www-east.elsevier.com/ida`.
2. Jaja J. *An Introduction to Parallel Algorithms*, Addison Wesley, (1992).
3. Jelonek J., Krawiec K. & Slowinski R. 'Rough set reduction of attributes and their domains for neural networks', *Computational Intelligence*, Vol. 11, No 2, (1995), pp. 339–347.
4. Kryszkiewicz M. & Rybinski H. 'Finding Reducts in Composed Information Systems', *Fundamenta Informaticae*, Vol. 2, No 2/3, (1996), pp. 183–196.
5. Orlowska M. & Orlowski M. 'Maintenance of Knowledge in Dynamic Information Systems', In: Slowinski R., (ed.), *Intelligent Decision Support. Handbook of Applications and Advances of the Rough Set Theory*, Kluwer Academic Publishers, Dordrecht, (1992), pp. 315–330.
6. Romanski S. 'Operations on Families of Sets for Exhaustive Search Given a Monotonic Function', In: Beeri, C., Smith, J.W., Dayal, U., (eds), *Proceedings of the 3rd International Conference on Data and Knowledge Bases*, Jerusalem, Israel, (1988), pp. 28–30.
7. Skowron A. & Rauszer C. 'The Discernibility Matrices and Functions in Information Systems', In: Slowinski R., (ed.), *Intelligent Decision Support. Handbook of Applications and Advances of the Rough Set Theory*, Kluwer Academic Publishers, Dordrecht, (1992), pp. 331–362.

8. Slowinski K., Stefanowski J., Antczak A. and Kwias Z. 'Rough Set Approach to the Verification of Indications for Treatment of Urinary Stones by Extracorporeal Shock Wave Lithotripsy (ESWL)', In: Lin T.Y., Wildberger A.M., (eds.), *Soft Computing*, Society for Computer Simulation, San Diego, California, (1995), pp. 142–145.
9. Ziarko W. & Shan N. 'Data-Based Acquisition and Incremental Modification of Classification Rules', *Computational Intelligence*, Vol. 11, No 2, (1995), pp. 357–370.