Covering with reducts - a fast algorithm for rule generation

Jakub Wróblewski

Institute of Mathematics
Warsaw University
Banacha 2, 02-097 Warsaw, Poland
e-mail: jakubw@jakubw.pl
http://www.jakubw.pl/about

Abstract. In a rough set approach to knowledge discovery problems, a set of rules is generated basing on training data using a notion of reduct. Because a problem of finding short reducts is NP-hard, we have to use several approximation techniques. A covering approach to the problem of generating rules based on information system is presented in this article. A new, efficient algorithm for finding local reducts for each object in data table is described, as well as its parallelization and some optimization notes. A problem of working with tolerances in our algorithm is discussed. Some experimental results generated on large data tables (concerned with real applications) are presented.

1 Introduction

Rough set expert systems base on the notion of a *reduct* ([7], [8]), a minimal subset of attributes which is sufficient to discern between objects with different decision values. A set of short reducts can be used to generate rules ([1]). A problem of short reducts generation is NP-hard, but an approximate algorithm (like the genetic one described in [9], [4] and implemented successfully - see [6]) can be used to obtain reducts in reasonable time. On the other hand, rules generated basing on reducts are often too specific and cannot classify new objects. Another types of reducts have been considered to improve efficiency on new objects (see [2]). One of the methods is to calculate reducts basing on a single object.

Let $A = (U, A \cup \{d\})$ be an *information system* (see [8]), where U - set of objects, A - set of attributes, d - decision.

Definition: A *local reduct* $R(o_i) \subseteq A$ (or a *reduct relative to decision and object* $o_i \in U$; o_i is called a *base object*) is a subset such that:

a) $\forall\ o_j \in U,\ d(o_i) \neq d(o_j) \Longrightarrow \exists\ a_k \in R$: $a_k(o_i) \neq a_k(o_j)$

b) R is minimal with respect to inclusion.

A classical reduct will be referred to as *global reduct*. A rule generated by a local reduct is concerned with the base object and may not recognize any other object from U. To assure that a set of rules will recognize (at least) all objects from the training set, we have to generate a local reduct for every object. A simple algorithm checking whether a subset is a local superreduct works at a

time complexity of $O(mn)$: we have to compare our base object with all other objects and check whether condition a) (see local reduct definition) holds. It takes $O(mn^2)$ time to do this for all objects (when we are looking for a local reduct for every object), where n = number of objects, m = number of attributes. This time complexity is not acceptable for large data tables. A fast approximation algorithm for local reducts generation is presented in the next section. In sections 3 and 4 some related topics, concerned with parallelization of the algorithm and dealing with tolerance are discussed. In section 5 some experimental results are presented.

2 Covering algorithm

An algorithm presented below realizes the following objective: assuming the information system is consistent, find a family of subsets R_1, R_2,... R_k such that for any object o_i from U at least one R_j is a local reduct (we will say, that R_j covers o_i). We will look for possibly small family R_1,... R_k, i.e. we will prefer these subsets which cover possibly many objects. We assume, that these subsets reflect regularities in data and generate more general rules - it means better classification of new samples and less memory required to store rules.

1. Let σ be a random permutation of attributes.
2. Let $R = A$ and N_1,... N_n - a table of numbers of local reducts found for each object. Set $N_j = 0$.
3. Test whether R is a local reduct for any object. If so, increment N_i for these objects and store rules.
4. Let $R = R - a_i$, where a_i - the first attribute from R. Calculate a number M_i of these objects, for which R is a (super)reduct, and which are not covered by reducts found previously. Let $R = R + a_i$.
5. Continue step 4. with the next attribute from R. Finish after collecting numbers M_i for all attributes.
6. Find the maximal number among M_i; if there are more than one such a number - get the first one with respect to the permutation σ. Let a_j - an attribute associated with this maximum. Let $R = R - a_j$.
7. Continue from 3. until R is empty.
8. If there is at least one uncovered object - let $R = A$, continue from 4.

Lemma. The algorithm described above generates a covering for all objects in at most $n = |U|$ cycles (by "cycle" we mean one sequence of steps from 2 to 8).

Proof. We will prove, that in one cycle at least one uncovered object is covered by newly produced reduct. When we find out, that a set R is a local superreduct for a number of objects not covered so far in step 6. of the algorithm, there are two possibilities: a) all M_i are equal to 0, but it means that R is a local reduct for all these objects (because it is a superreduct and none of its subsets is a superreduct) so we have covered some new objects; b) there exists an $M_i > 0$, i.e. at least one subset of R is a superreduct for M_i objects not covered so far -

we continue from step 4. If our subset R has two attributes, possibility b) means, that there exists a local reduct with one attribute (a superreduct with only one attribute must be a reduct). So, in one cycle (starting from $R = A$, which is a local superreduct for all objects) we either realize possibility a) or, in the worst case, achieve a reduct containing only one attribute.

We need to have a method of determining whether a subset is a superreduct to complete our algorithm.

1. Sort a table of objects using attribute values (for attributes belonging to R).
2. Scan the table of sorted objects one by one. Our objects are divided into groups with equal values on attributes (abstract classes of indiscernibility relation generated by R, see [8]).
3. If a group has an uniform value of decision - it means that R is a local superreduct for objects belonging to this group. If not - R is not a local superreduct for these ones.

Since we may use a fast method of sorting, our algorithm has the complexity of $mn\ log\,(n)$, where n = number of objects, m = number of attributes.

3 Parallel algorithm and practical notes

The algorithm described in the previous section covers all objects by at least one reduct. On the other hand, the more reducts for each object we find, the more rules we can generate. Since the algorithm is deterministic for a given permutation σ, we have the following possibilities:

1. We may choose a set of p permutations $\sigma_1, \ldots\ \sigma_p$ and generate p coverings using this algorithm in parallel on p machines. When permutations are different, the obtained coverings usually are different too.
2. We may do the same on one machine in sequential way. In this case we can perform an additional optimization: at each stage of algorithm we look for covering for only these objects which are covered in minimal degree during the previous stages.
3. We may use an evolutionary algorithm to find the best permutation - i.e. the permutation generating a covering using a minimal set of possibly short reducts. An order-based genetic algorithm (see [3], [10]) can be used in case of sequential as well as parallel computations.

When we check whether a subset R is a local superreduct for any object, we can easily check whether it is a global superreduct (R is a global superreduct $\Longleftrightarrow$ R is a local superreduct for all objects). On the other hand, the algorithm of finding global reducts described in [4] and [11] uses a structure called "reduct store" containing all known global superreducts of information system. Thus, we can check whether R is known as a global superreduct before we start to sort

our object set, as well as we can add R to this structure when we find out that R is a global superreduct. Moreover, the same structure can be used by many agents in parallel implementation (each agent calculates covering for different permutation) and by one specialized agent calculating global reducts.

4 Tolerance

We use local reducts to generate rules which are more general than these generated basing on general reducts. On the other hand, these rules may still be too specific - especially when we work on numerical data. One of the ways to manage this problem is to use a discretization technique (see e.g. [5]). Alternatively, we can use a *tolerance measure*, which allows us to treat two different (but close) values as equal.

An algorithm presented in section 2. can be easily adopted to this new situation, in case a tolerance relation is transitive. In this case we can sort a set of objects and divide it into classes of this relation - then continue with the standard algorithm. Alternatively, we can initially replace attributes‘ values with their representatives (found by e.g. methods of scalar quantization or discretization).

Unfortunately, many tolerance relations are not transitive, and we cannot simply sort data and check adjacent pairs of objects. More research is needed to use our algorithm in this case.

5 Experimental results

The algorithm described in section 2. was implemented and tested on several information systems used in real applications - results are shown in the table presented below.

Size: obj × attr	#red	Time [sec]
4,492 × 36	1	13
	10	49
24,000 × 10	1	25
22,000 × 27	1	90
	5	1600
47,000 × 28	1	360

Calculations were performed on Pentium-200 machine. The first data set is the "Satellite image" database, the second is the "Shuttle" database. The column "#red" indicates how many reducts (at least) we found for each object.

The results show, that our new method is relatively fast, even for large data tables (finding local reducts for each object using the previous methods takes many hours for tables with number of objects greater than 20,000), especially when we are interested in just a covering of objects. Actually, when we cover objects by at least one reduct, an average number of reducts covering an object is usually equal to about 3.5.

6 Conclusions and future work

We have presented a covering approach to rule generation problem and an efficient algorithm for finding local reducts for a set of objects. A computation time for this algorithm is close to the time of global reduct finding ([4]). Our new method is fast, and it should generate more general rules - a comparison of efficiency of these rules generated in classical and a new way will be performed in the future. Another direction of future research is to implement and test an evolutionary algorithm (see section 3.) and a tolerance-based techniques. Moreover, a parallel system has been not implemented so far.

Acknowledgement

This work was supported by Polish State Committee for Scientific Research grant #8T11C01011 and Research Program of European Union - ESPRIT-CRIT2 No. 20288

References

1. Bazan J., Skowron A., Synak P., 1994. *Dynamic reducts as a tool for extracting laws from decision tables*, Proc. of the Symp. on Methodologies for Intelligent Systems, Charlotte, NC, October 16-19, 1994, Lecture Notes in Artificial Intelligence 869, Springer-Verlag, Berlin 1994, 346-355, also in: ICS Research Report 43/94, Warsaw University of Technology.
2. Bazan J., 1998. *A Comparison of Dynamic and non-Dynamic Rough Set Methods for Extracting Laws from Decision Tables.* In: L. Polkowski, A. Skowron (eds.). Rough Sets in Knowledge Discovery. Physica Verlag, 1998.
3. Goldberg D.E., 1989. *GA in Search, Optimisation, and Machine Learning.* Addison-Wesley.
4. Nguyen S. H., Skowron A., Synak P., Wróblewski J., 1997. *Knowledge Discovery in Databases: Rough Set Approach.* Proc. of The Seventh International Fuzzy Systems Association World Congress, vol. II, pp. 204-209, IFSA97, Prague, Czech Republic.
5. Nguyen H. S., Nguyen S. H., 1998. *Discretization Methods in Data Mining.* In: L. Polkowski, A. Skowron (eds.). Rough Sets in Knowledge Discovery. Physica Verlag, 1998.
6. Øhrn A., Komorowski J., 1997. *Rosetta - A rough set toolkit for analysis of data.* Proc. of Third International Join Conference on Information Sciences (JCIS97), Durham, NC, USA, March 1 - 5, 3 (1997), pp. 403-407.
7. Pawlak Z., 1991. *Rough sets: Theoretical aspects of reasoning about data.* Kluwer: Dordrecht 1991.
8. Skowron A., Rauszer C., 1992. *The Discernibility Matrices and Functions in Information Systems.* In: R. Slowiński (ed.): Intelligent Decision Support. Handbook of Applications and Advances of the Rough Sets Theory. Kluwer: Dordrecht 1992, pp: 331 - 362.
9. Wróblewski J., 1995. *Finding minimal reducts using genetic algorithms.* Proc. of the Second Annual Join Conference on Information Sciences, pp.186-189, September 28-October 1, 1995, Wrightsville Beach, NC. Also in: ICS Research report 16/95, Warsaw University of Technology.

10. Wróblewski J., 1996. *Theoretical Foundations of Order-Based Genetic Algorithms*. Fundamenta Informaticae, vol. 28 (3, 4), pp: 423-430. IOS Press, 1996.
11. Wróblewski J., 1998. *Genetic algorithms in decomposition and classification problem*. In: L. Polkowski, A. Skowron (eds.). Rough Sets in Knowledge Discovery. Physica Verlag, 1998.