Practical Applications of Genetic Algorithms for Efficient Reduct Computation

Anders Torvill Bjorvand	Jan Komorowski
Troll Data Inc.	Department of Computer and Information Science
P.O. Box 335	The Norwegian University of Science and Technology
N-1801 Askim, Norway	N-7034 Trondheim, Norway
torvill@trolldata.no	Jan.Komorowski@idi.ntnu.no

Keywords: Rough sets, genetic algorithms, data reduction.

ABSTRACT

Large collections of data may contain valuable but undiscovered knowledge. The process of making such knowledge explicit is called knowledge discovery. The rough set theory, introduced by Pawlak and his colleagues (see eg [4], [5] or [7]), has been successfully applied to many problems in this area.

A real problem when taking rough sets from theory to application is in the analysis of large data sets because the complexity is very high. The essential task of finding reducts is reported NP-hard in [6]. Many other operations related to the rough set theory also have a high complexity, but this paper focuses on computational methods of approximating reducts. In particular, an application of genetic algorithms will be discussed.

The use of genetic algorithms for finding reducts was introduced in [8]. We implemented the "Rough Enough" software system which applies genetic algorithms for computing reducts ([1] and [2]). The techniques presented in [8] have been a foundation, but several variations and practical improvements have been implemented. This work led to the discovery of many practical details and possible pitfalls when implementing such techniques, and these discoveries form the focus of this paper. Using the experience from the "Rough Enough" implementation, this paper discusses some rules of thumb for the implementation of reduct computations with genetic algorithms.

INTRODUCTION

This paper introduces some genetic algorithms that have been used in experiments in eg [1]. We will also emphasize some practical traps and pitfalls that may easily distinguish between success and failure. Based on the information given in this paper, the reader will hopefully be better equipped when designing a rough set system of his own.

Due to restrictions on the length of this paper, we have to assume that the reader has a working knowledge of both rough set theory and genetic algorithms. For an introduction to these topics see, for instance, [3],[5] and [7].

There are two general and equally critical issues in any implementation of genetic algorithms:

- representation
- evaluation criteria – more often called fitness functions.

The representation is the data structure we use to represent our population of candidate solutions, in our particular case – reduct candidates. The fitness function is a quantitative measure of the fitness of the individuals in our population – ie whether or not they solve our problem.

We will concentrate on only one representation and experiment further with the fitness function. We will emphasize the importance of starting with a good initial population, and we will explain techniques to avoid ”dead ends” in the evolution.

REPRESENTATION

The discernibility matrix (see [6]) is used as a basis for computing reducts. A special representation of the discernibility matrix is used in order to utilize fast bitwise AND-instructions available in the CPU of any general purpose computer. This is illustrated with a simple hands-on example:

Let us use an information system describing some dogs:

Object number	Breed	Tail	Hair	Size
1	Doberman Pincher	Short	Short	Big
2	Dalmatian	Long	Short	Big
3	Cocker Spaniel	Short	Long	Small
4	German Shepherd	Long	Medium	Big

We wish to discern between the dogs using the three last features. The attribute describing the breed is masked. The indices of the discernibility matrix below show the attributes that can distinguish between the two corresponding objects:

Objects/Objects	1	2	3	4
1	∅			
2	{Tail}	∅		
3	{Hair, Size}	{Tail, Hair, Size}	∅	
4	{Tail, Hair}	{Hair}	{Tail, Hair, Size}	∅

We want to obtain a form of the discernibility matrix more suitable for binary manipulation. For every index of the discernibility matrix (every pair of object combination), we use a binary array with as many elements as the number of attributes. A binary one in this array indicates that this particular attribute discerns between these two objects. A zero, of course, indicates the opposite. In this example, the order of the attributes in the binary array is: Tail, Hair, Size:

Object Combination	Attribute array
1,2	100
1,3	011
1,4	110
2,3	111
2,4	010
3,4	111

Based on the above representation of the discernibility matrix, a representation of our reduct candidates - the individuals in our population - is constructed. A reduct is represented as a binary array where every binary element represents an attribute of the information system. If an element is equal to 1, that particular attribute is part of the reduct candidate. Otherwise it is not.

HOW TO OBTAIN REDUCTS

There are two requirements for a reduct candidate to really be a reduct:

- It must be able to discern between all our objects. If a bitwise AND is performed with a result different from all zeros between our candidate and all members of our discernibility matrix representation, this is fullfilled.
- There must not exist another candidate whose set of attributes is both a true subset of our candidates' attributes and is at the same time able to discern between all objects.

Deterministic Solution

The deterministic algorithm for finding the set of reducts can be described with the following pseudocode:

```
set_of_reducts ← ∅
for i from 1 to number_of_attributes
        for all reduct candidates, reduct_candidate, containing i number of attributes
                if (reduct_candidate discerns between all objects) and
                  (no member of set_of_reducts is a true subset of reduct_candidate) then
                        add reduct_candidate to set_of_reducts
                endif
        endfor
endfor
```

Running this algorithm on the dog-example, the following reduct candidates are evaluated:

Reduct candidate	Comment
000	Does not discern between any
100	Satisfies 4 out of the 6 arrays in our discernibility matrix
010	Satisfies 5 out of the 6 arrays in our discernibility matrix
001	Satisfies 3 out of the 6 arrays in our discernibility matrix
101	Satisfies 5 out of the 6 arrays in our discernibility matrix
110	Satisfies 6 out of the 6 arrays in our discernibility matrix - reduct
011	Satisfies 5 out of the 6 arrays in our discernibility matrix
111	Satisfies 6 out of the 6 arrays in our discernibility matrix, but the reduct 110 is a true subset - not a reduct

So our only reduct is 110 or {Tail, Hair}.

Approximation by a Genetic Algorithm

The deterministic algorithm described in the previous section was shown to be NP-hard in [6]. This motivates us to search for suitable approximations with lower computational complexity. Genetic algorithms were reported successful for this particular application in [8].
We will here show the algorithm used in [1]. Unfortunately, the lack of space does not allow us to elaborate and give motivation for all the details of our fitness function. The reader is referred to [1]. The fitness function is as follows:

$$F(\vec{v}) = \frac{N - L_{\vec{v}}}{N} + \frac{C_{\vec{v}}}{(m^2 - m)/2}$$

$\vec{v} = reduct\ candidate$

$N = number\ of\ available\ attributes$

$L_{\vec{v}} = number\ of\ 1's\ in\ \vec{v}$

$C_{\vec{v}} = number\ of\ object\ combinations\ that\ \vec{v}\ can\ discern\ between$

$m = number\ of\ objects$

The fitness function consists of a summation of two parts. The first part gives the candidate credit for being short (few 1's). The second part determines to what extent the candidate can discern between the objects (a reduct can discern between all objects). The displayed fitness function is extremely simple and is only useful for explanatory purposes because of its simplicity. More advanced variations can be found in [1], [2] and [8].

Every individual in the current generation is evaluated with the fitness function. They are then given a probability for breeding according to their fitness value. The more fit an individual is, the more chromosomes they pass on to the next generation through breeding.

POSSIBLE PROBLEMS WHEN UTILIZING GENETIC ALGORITHMS

One of the key benefits of using genetic algorithms is that a problem can be solved without a very detailed understanding of the basic problem. One does not even need to know how to give an exact solution to the problem.
Genetic algorithms will often give a fairly good approximation to a problem with very little effort. However, there is considerable potential for making the process more effective both in terms of speed and the quality of approximation. This is accomplished by applying domain knowledge. The techniques/hints presented here are based on the work reported in [1]. We will stress the following factors:

- The importance of a good initial poulation - and how to obtain it.
- How to avoid "dead ends" - ie how to avoid wasting much processing power in a direction of search that can be described as a blind alley.

These two points were discovered to be important when implementing the "Rough Enough" software system described in [1] and [2]. Many other, possibly undiscovered, factors could, of course, be equally important.

How to Obtain a Good Initial Population

The closer the initial population is to the final result, the faster we will get there. How can we provide an initial population with an overall fitness that is better than a random selection? There are many simple heuristics, so we mention a few:

The core, if it exists, should be present as one of the candidates. The attributes from the core should also be included in all the other candidates.

The number of 1's in the candidates should be similar to the number of 1's in the reducts. This is however very hard to know in advance. We have experienced experiments where the number of 1's

in the reducts has been approximately 10% of the number of attributes. In these cases, it has been very unfortunate to let the number of 1's in the initial population average around 50% of the total number. In another experiment, the reducts contained approximately 50% of the attributes. In that case, the best results were obtained by letting the average number of attributes in the reduct candidates be 50%. Since it is difficult to predict reduct length in advance, we suggest the following functionality: Allow the input to the algorithm to determine the average size of the reducts in case the user should have sufficient domain knowledge. Otherwise, see to it that the variation from the average is considerable.

The expert very often has domain knowledge about "sure bets" - attributes that he is relatively certain belong to most reducts. This can be provided by allowing the user to assign relative weight to each attribute when creating the initial population. In cases where the core is non-empty, we do not need this information to the same extent. However, the core in real world data is quite rare. When data is collected from different sources, which is often the case, it is common to have a set of reducts whose disjunction is the empty set.

The attributes should be weighted according to how many objects they can discern. Continuing our earlier example, the relative weights would become the following:

Attribute number	Relative weight
1	4/6
2	4/6
3	4/6

In this particular case, weighting would be of no use, but the point would none the less be the same.

How to Avoid "Dead Ends"

The generations produced by a genetic algorithm can easily enter a "dead end" using much computational power in the wrong search direction. When such a situation is about to occur, the system must detect this and force the search in another, hopefully more fruitful direction. This is normally accomplished by setting a proper constant mutation rate, preventing all individuals in the population from becoming equal. Experiments described in [1] showed that if this should work, the mutation rate had to change dynamically. We have therefore implemented a mutation rate that is proportional to the redundancy in the population. In other words, when the individuals of the population are getting too close to one another ie they are relatives, the mutation rate increases to produce a greater diversity (somewhat related to the biological phenomenon of inbreeding, but certainly with a better result).

CONCLUSION

Genetic algorithms have proved effective in calculating reducts. Experiments taking several months by traditional methods can be calculated within an hour using the techniques described in this paper. Applying and understanding such techniques is crucial for doing research and experiments on large datasets.

FUTURE RESEARCH

There is still a considerable potential for making genetic algorithms more effective. Further experiments should be conducted and do in fact constitute the most important way to discover new and better ways of designing these algorithms.
We think that much work should be done to provide good initial populations by means of computationally inexpensive methods.

REFERENCES

[1] Bjorvand, Anders Torvill (1996). *Time Series and Rough Sets*. Master's Thesis, the Norwegian Institute of Technology, Department of Computer Systems, Trondheim, Norway.

[2] Bjorvand, Anders Torvill (1997). *'Rough Enough' - A System Supporting the Rough Sets Approach.* Sixth Scandinavian Conference on Artificial Intelligence 1997. Helsinki, Finland. To appear.

[3] Michalewicz, Zbigniew (1992). *Genetic Algorithms + Data Structures = Evolution Programs*. Springer Verlag.

[4] Pawlak, Zdzislaw (1982). "Rough Sets". Printed in *International Journal of Computer and Information Sciences*, **11**, pp. 341-356.

[5] Pawlak, Zdzislaw (1991). *Rough Sets - Theoretical Aspects of Reasoning About Data*. Kluwer Academic Publishers, Dordrecht.

[6] Skowron, Andrzej and Rauszer, Cecylia (1992). "The Discernibility Matrices and Functions in Information Systems". Printed in *Intelligent Decision Support - Handbook of Applications and Advances of the Rough Sets Theory*. Edited by Roman Slowinski. Pp. 331-362. Kluwer Academic Publishers, Dordrecht.

[7] Slowinski, Roman (1992) editor. *Intelligent Decision Support - Handbook of Applications and Advances of the Rough Sets Theory*. Kluwer Academic Publishers, Dordrecht.

[8] Wróblewski, Jakub (1995). *Finding Minimal Reducts Using Genetic Algorithms (extended version)*. Warsaw University of Technology - Institute of Computer Science - Reports - 16/95.