# A Reduct Solving Parallel Algorithm Based on Relational Extension Matrix

Pei Li Zhou
Faculty of Information Technology
Monash University
Melbourne Australia
joezhou@mail1.monash.edu.au

Salahadin Mohammed
Dept. of Info. & Comp. Sci.
KFUPM
Dhahran, Saudi Arabia
adam@kfupm.edu.sa

**Abstract:** In this paper, on the basis of studying the limitations of the basic rough set model, we present Tolerance Information Systems, which is based on a family set of tolerance relations between objects when given a set of tolerance relations. The model inherits most of the characteristics of the basic model of rough set; and they also have a better effect of approximation classification. Based on this model, we propose two algorithms that will give us one near-optimal attributes reduct in memory and process efficient way, the first one is a single processor algorithm which uses concepts from relation and extension matrices. Based on the first algorithm, its parallel processor version is proposed, the parallel version is far more efficient.

**Keywords:** Machine Learning, Rough Set, Reduct, Information System, Extension Matrix, Relation Matrix, Tolerance Relation, Knowledge Acquisition, Parallel, Knowledge Representation

## 1. Introduction

Rough set theory has some advantages over other similar formal tools, and has been used widely in some areas **[1, 2, 3, 4],** such as knowledge acquisition, machine learning, knowledge representation. But there are some limitations in its application. Firstly, when we could only do partial classification, the classification done by the rough set model is completely correct and certain; it could not give a classification with a kind of controllable misclassification. However, in a real time situation, this kind of classification could give better understanding and processing of the analyzed data. Secondly, another limitation of approximation space is from the view point which the universe U of the considering data objects are all known, and the derived result from that model is only usable for that object set. However, in real life situations, we obviously need to extend the result derived from the limited object set to a larger data set. To solve this problem better, we here present one extended rough set models; that is based on a family set of tolerance relations between objects when given a set of tolerance relations.

## 2. Tolerance Information Systems

Based on the definition of approximation space **K** = ($U$, **R**, $\tau$)**[4, 5, 6]**, we first could give the definition of a rather restricted special tolerance information system. In the process of mapping $\tau$ from the corresponding information system, we could get rid of the restriction, to extend it to general tolerance information system.

**Definition 1** One special tolerance information system is a triple **S** = ($U$, $A$, $\tau$), where $U = \{x_1, x_2, \ldots, x_n\}$ is known as the non-empty finite objects set, which called Universe. $A$ is the non-empty finite set of primitive attributes $a_i$ ($i$=1, 2, …,$k$). The mapping $\tau$ is the mapping from **powerset**($A$)-$\{\varnothing, \{a_1\}, \{a_2\}, \ldots, \{a_k\}\}$ into the family set **TS**(**S**) of tolerance relations on universe $U$. It satisfies the following characteristics: Every primitive attribute $a_i \in A$ is a total function, which is defined on $X_\mathrm{i} \subseteq U$ ($i$=1, 2, …, $k$), i.e. $a_i : \mathrm{X_i} \rightarrow \mathrm{V_{ai}}$, where $V_{ai}$ is the value domain of the primitive attribute $a_i$ ; for $x \in U - X_i$ , the function $a_i$ have no definition on it. Every primitive attribute $a_i \in A$ is corresponding to a tolerance relation $I\{a_i\}$ defined on $X_i \subseteq U$ ($i$=1, 2,…,$k$), i.e. To any attributes subset $B \in$ **powerset**($A$)-$\{\varnothing, \{a_1\}, \{a_2\}, \ldots, \{a_k\}\}$, there exists one related definition on universe $U$ which is a binary relation $\tau(B) = I_B \in$ **TR**(**S**), regarded as the tolerance relation of the attribute subset $B$, where it satisfies the following properties:

a) Monotony of the differences between information vectors: $\forall\ x_1, x_2, y_1, y_2 \in INF(B)$, if $((y_2\text{-}y_1) \cup (y_1\text{-}y_2)) \subseteq ((x_2\text{-}x_1) \cup (x_1\text{-}x_2))$, then $(y_1 I_B y_2) \rightarrow (x_1 I_B x_2)$;

b) Decreasing monotony of the tolerance attributes set: $\forall\ B \subseteq C$, and $\forall\ x, y \in INF(C)$, we have $((x \mid C)\ I_C\ (y \mid C)) \rightarrow ((x \mid B)\ I_B\ (y \mid B))$.

In tolerance information system **S** = ($U$, $A$, $\tau$), we have boarded the limitation to every primitive attribute. On one side we allow some primitive attributes of some objects to have missing values, on the other side we boarded the equivalence relations from the basic information system into the tolerance relations. Thus, it becomes closer to the original description of primitive data table. But, because we want to generate a tolerance relation of all objects on universe $U$ based on one attributes subset $B \subseteq A$, so when study the specific definition of the mapping $\tau$, we must give the processing strategies of missing attribute values. Note, the reason why we ask for the tolerance relation to be defined on the universe $U$, is its prime purpose is to do reduction of attributes. For the 2nd item of the definition above, sometimes we need to extend the definition of tolerance relation $\mathrm{I}_{ai}$ from objects set $X_i$ into the universe $U$, then it is noted as $\mathrm{I}'_{ai}$. This requires us to give the proper strategies to process the missing attribute values.

The monotony of the information vectors difference is referred to two information vectors on attributes set B. If two information vectors with less attribute values difference are non-tolerant, and these attribute values difference is contained in another two information vectors with more attribute values difference. Then the information vectors with more attribute values difference are also non-tolerant. The monotony of the tolerance attribute set in fact implies that every attribute has the same kind of importance in assumption information system; it is a rather restricted limitation. The monotony of the tolerance attributes set in the special tolerance information system actually pays more attention to the equivalent importance of all attributes. In some real world implementations, this criterion may not be satisfied, and we will now define the general tolerance information system below.

**Definition 2** One general tolerance information system is a triple **S**=(U, A, $\tau$), where U=$\{x_1, x_2, \ldots, x_n\}$ is the non-empty finite objects set known as Universe. A is the non-empty finite set of primitive attributes $a_i$ ( $i$= 1, 2,…, k). The mapping $\tau$ is the mapping from **powerset** (A) -$\{\varnothing, \{a_1\}, \{a_2\}, \ldots, \{a_k\}\}$ into the family set **TS**(**S**) of tolerance relations on universe U. It satisfies the following characteristics: Every primitive attribute

$a_i \in$ A is a total function, which is defined on $X_i \subseteq$ U (i=1, 2, …, $k$), i.e. $a_i$: $X_i \rightarrow V_{ai}$, where $V_{ai}$ is the value domain of the primitive attribute $a_i$ ; for $x \in$ U- $X_i$, the function $a_i$ has no definition on it. Every primitive attribute $a_i \in$ A is corresponds to a tolerance relation $Ia_i$ defined on X⊆U ($i$=1, 2, …, $k$). To any attributes subset B ∈ **Powerset**(A)-{ ∅ , {$a_1$} , {$a_2$} ,…, {$a_k$}}, there exists one related definition on universe U which is a binary relation $\tau(B)=I_B \in$ **TR**(**S**), regarded as the tolerance relation of the attribute subset B, where it satisfies the following properties:

(1) Monotony of the differences between information vectors: $\forall$ $x_1$, $x_2$, $y_1$, $y_2$ ∈ INF(B), if $((y_2-y_1) \cup (y_1-y_2)) \subseteq ((x_2-x_1) \cup (x_1-x_2))$, then $(y_1\ I_B\ y_2) \rightarrow (x_1\ I_B\ x_2)$;

(2) Increasing monotony of the tolerance attributes set: $\forall$ B ⊆ C, and $\forall$ x, y ∈ INF(C), we have (x, y) $\in I_B \rightarrow$ (x, y) $\in I_C$, and (x, y) $\neg \in I_B \rightarrow$ (x, y) $\neg \in I_C$.

The increasing monotony of tolerance attributes set means that to every two objects x and y, if they are tolerant (or non-tolerant) on a smaller attributes set, then they are also tolerant (non-tolerant) on the attributes set which contains that attributes set. Generally in real world implementations, this criterion can be satisfied. But note that the size of this kind of small attributes set has certain requirements; this attributes set must contain the minimum attributes set which is required by the tolerance of the two objects. Furthermore, note that in special tolerance information systems, it asks for the decreasing monotony of tolerance attributes set, but here this definition asks for the increasing monotony of tolerance attributes set. We must point out that in the special tolerance information system, satisfying decreasing monotony of tolerance attributes set will give rise to the required increasing monotony of tolerance attributes set, that is $\forall$ B ⊆ C and $\forall$ x, y ∈ INF(C), (x, y) $\neq \in I_B \rightarrow$ (x, y) $\neq \in I_C$. We use this monotony to solve the problems of reduction of attributes. In later sections when we refer to attributes monotony, it just meant the decreasing monotony of the particular tolerance attributes set. To define the specific tolerance relation between objects normally requires extra domain knowledge. For example, in one of the pattern recognition problems, having 3 major characteristics {$a$, $b$, $c$} and 5 minor characteristics {$d$, $e$, $f$, $g$, $h$} to describe recognized objects. The tolerance of two objects on the characteristics set A={$a$, $b$, $c$, $d$, $e$, $f$, $g$, $h$} is defined as: if there are at least 2 major characteristics same, or one major characteristic and at least 4 minor characteristics same, then these two objects are tolerant. Let is be that known object $x_1$ and $x_2$ are tolerant on attributes set B={$a$, $d$, $e$, $f$, $g$}, if get rid of attribute $a$, then object $x_1$ and $x_2$ are non-tolerant on attributes set {$d$, $e$, $f$, $g$}.

## 3. Description of the Algorithm Based on Extension and Relation Matrices

In this section, we will give out the algorithm which solves a near optimal relative tolerance reduct. It has three characteristics; first of all it uses tolerance information systems proposed in the above session; secondly it uses the concepts of positive and negative examples (negative extension matrix) from extension matrix [**7**, **8, 9**]; at last it uses relation matrix for memory storage and processing.

### 3.1 The algorithm to solve the near optimal relative tolerance reduct

Given a tolerance decision information system **S** = ($U$, $A$, {$d$}, $\tau$), we propose the algorithm below to solve the near optimal relative tolerance reduct R of attribute set A.

Algorithm 3.1.1 the algorithm to solve the near optimal relative tolerance reduct based on tolerance extension matrix.

**I**nput: A given tolerance decision information system **S** = (*U*, *A*, {*d*}, τ);

**O**utput: one near optimal relative tolerance reduct **R** of attribute set A;

**B**egin

**D**ivide all objects (*U*) into positive and negative examples, according decision attribute (*d*) from the given tolerance decision information system **S** = (*U*, *A*, {*d*}, τ);

**R** = ∅;

**C**onstruct all Negative Extension Matrices from all positive examples;

**S**tore all Negative Extension Matrices in memory with the structure of Relational Extension Matrices.

**R**educt Matrix **M**= ∩ all Negative Extension Matrix stored in Relation Matrices;

(Negative Extension Matrix 1 ∩ Negative Extension Matrix 2 ∩ Negative Extension Matrix 2 ∩ Negative Extension Matrix 3 ∩ Negative Extension Matrix *n*)

**R** = Remaining attributes that still contain value “1” in **M;**

Output Reduct **R**;

**E**nd
The above are the steps of the algorithm. Now we give out an example using the above algorithm.

**3.2 An example of the above algorithm.**

Example below uses the decision table provided by paper **[6]** (table 3.1) to explain the process/steps of algorithm 3.1.1.

| | **H**eight | **W**eight | hai**R** | **E**yes | D |
|---|---|---|---|---|---|
| 1 | Short | Light | Dark | Blue | 1 |
| 2 | Tall | Heavy | Dark | Blue | 1 |
| 3 | Tall | Heavy | Dark | Brown | 1 |
| 4 | Tall | Heavy | Blond | Brown | 1 |
| 5 | Short | Light | Blond | Brown | 1 |
| 6 | Tall | Heavy | Red | Blue | 2 |
| 7 | Short | Light | Blond | Blue | 2 |
| 8 | Tall | Heavy | Blond | Blue | 2 |

Table 3.1 The decision table of a tolerance information system **S**

We could now construct the tolerance decision information system based on the decision table above. It is **S** = (*U*, *A*, {*d*}, τ), where *U* = {1,2,3,4,5,6,7,8}; condition attribute set *A* = {*H,W,R,E*}, where attributes are **H**eight, **W**eight, hai**R**, **E**yes, they all generate the equivalence relation on universe *U*; decision attribute is *d*.

Mapping τ Is defined as: ∀*B* ⊆ *A*, where *card*(*B*)≥1, if ∀*x,y* ∊ *U* on attribute subset *B* have values different on one attribute, then these two objects are non-tolerant, otherwise they are tolerant.

We then divide *U* into positive and negative examples according to the decision attribute *d*. We have positive examples {1,2,3,4,5} and negative examples {6,7,8}. So we now ready to construct negative extension matrices for the positives examples and derive reduct matrix **M** from all negative extension matrices.

Step 1:

Negative Extension Matrix 1:

| | **H**eight | **W**eight | hai**R** | **E**yes |
|---|---|---|---|---|
| 1 | Short | Light | Dark | Void |
| 2 | Void | Void | Dark | Void |
| 3 | Void | Void | Dark | Brown |
| 4 | Void | Void | Blond | Brown |
| 5 | Short | Light | Blond | Brown |

Constructed for Positive Example 1:

[ Tall Heavy Red Blue ]

Step 2:

Note, please here onwards, we will denote ‘Void’ as ‘*’.

Negative Extension Matrix 2:

| | **H**eight | **W**eight | hai**R** | **E**yes |
|---|---|---|---|---|
| 1 | * | * | Dark | * |
| 2 | Tall | Heavy | Dark | * |
| 3 | Tall | Heavy | Dark | Brown |
| 4 | Tall | Heavy | * | Brown |
| 5 | * | * | * | Brown |

Constructed for Positive Example 2:

[ Short Light Blond Blue ]

Step 3:

Negative Extension Matrix 3:

| | **H**eight | **W**eight | hai**R** | **E**yes |
|---|---|---|---|---|
| 1 | Short | Light | Dark | * |
| 2 | * | * | Dark | * |
| 3 | * | * | Dark | Brown |
| 4 | * | * | * | Brown |
| 5 | * | Light | * | Brown |

Constructed for Positive Example 3:

[ Tall Heavy Blond Blue ]

Step 4:

Negative Extension Matrix 1 ∩ Negative Extension Matrix 2:

| | **H**eight | **W**eight | hai**R** | **E**yes |
|---|---|---|---|---|
| 1 | * | * | Dark | * |
| 2 | * | * | Dark | * |
| 3 | * | * | Dark | Brown |
| 4 | * | * | * | Brown |
| 5 | * | * | * | Brown |

Step 5:

Negative Extension Matrix 1 ∩ Negative Extension Matrix 2 ∩ Negative Extension Matrix 3:

| | **H**eight | **W**eight | hai**R** | **E**yes |
|---|---|---|---|---|
| 1 | * | * | Dark | * |
| 2 | * | * | Dark | * |
| 3 | * | * | Dark | Brown |
| 4 | * | * | * | Brown |
| 5 | * | * | * | Brown |

The above five steps illustrate how negative extension matrices are constructed. We will not use the original extension matrix format here, because of three reasons:

1) The extension matrix is memory hungry even with values of “void”, they still have been allocated memory for in such data structure; in our proposed relational extension matrix, we could use ‘0’ to represent “void”.

2) Our aim of construct extension matrices here is to study the relationship between objects/attributes, not their attribute values, so the exact values are not important to us, thus we could use ‘1’ to represent a relationship exists in our relational extension matrix.

3) It takes a lot of time to perform value comparisons during the ∩ operations. With the introduction of relational extension matrix, ∩ operations are just much easy to perform and natural to digital computers.

Thus we could repeat the above five steps using our relational extension matrices below. It is memory and process efficient due to the uses of ‘1’ / ‘0’ only in the relational extension matrix, it is very efficient to store in the memory (integer only) and efficient in processing as well (easy to construct and quick to compare).

Step 1:

Relational Extension Matrix 1:

| | Height | Weight | Hair | Eyes |
|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 0 |
| 2 | 0 | 0 | 1 | 0 |
| 3 | 0 | 0 | 1 | 1 |
| 4 | 0 | 0 | 1 | 1 |
| 5 | 1 | 1 | 1 | 1 |

Constructed for Positive Example 1:

[ Tall Heavy Red Blue ]

Step 2:

Relational Extension Matrix 2:

| | Height | Weight | Hair | Eyes |
|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 0 |
| 2 | 1 | 1 | 1 | 0 |
| 3 | 1 | 1 | 1 | 1 |
| 4 | 1 | 1 | 0 | 1 |
| 5 | 0 | 0 | 0 | 1 |

Constructed for Positive Example 2:

[ Short Light Blond Blue ]

Step 3:

Relational Extension Matrix 3:

| | Height | Weight | Hair | Eyes |
|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 0 |
| 2 | 0 | 0 | 1 | 0 |
| 3 | 0 | 0 | 1 | 1 |
| 4 | 0 | 0 | 0 | 1 |
| 5 | 0 | 1 | 0 | 1 |

Constructed for Positive Example 3:

[ Tall Heavy Blond Blue ]

Step 4:

Relational Extension Matrix 1 ∩ Extension Matrix 2:

| | Height | Weight | Hair | Eyes |
|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 0 |
| 2 | 0 | 0 | 1 | 0 |
| 3 | 0 | 0 | 1 | 1 |
| 4 | 0 | 0 | 0 | 1 |
| 5 | 0 | 0 | 0 | 1 |

Step 5:

Relational Extension Matrix 1 ∩ Relational Extension Matrix 2 ∩ Relational Extension Matrix 3:

| | Height | Weight | Hair | Eyes |
|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 0 |
| 2 | 0 | 0 | 1 | 0 |
| 3 | **0** | **0** | **1** | **1** |
| 4 | 0 | 0 | 0 | 1 |
| 5 | 0 | 0 | 0 | 1 |

This above relational extension matrix is the Reduct Matrix **M** we illustrated in algorithm 3.1. Take out the remaining attributes whose values are “1”s, then we got two attributes: **Hair** and **Eyes**.

We could tell that our near optimal reduct (Hair, Eyes) is in this case the optimal reduct that could be solved [**10**]. Please note that our algorithm may not always give the optimal reduct, but it will certainly contain one optimal reduct, so we always will have a near optimal reduct. Please note for space reason we have used a small table, so in this example, our tolerance relations are the same as equivalence relations. For uses of tolerance relations in tolerance information systems to solve reducts, refer to reference [**11**].

## 4. The Proposed Parallel Algorithm

In algorithm 3.1.1 discussed in Section 3.1, we can see that each $a_i$ value in a positive example is matched with all the $a_i$ values of the negative examples. Our proposed algorithm exploits this property to do the same computation in parallel in a mesh of $m \times n$ processors. In the mesh, a processor in row $i$ and column $j$ is denoted as $p_{i,j}$, for $0 \leq i < m$ and for $0 \leq j < n$.

The algorithm starts by partitioning $U$ row wise into *u* and *v*, where *u* consists of all the positive examples and *v* consists of all the negative examples of U. Let |u| and |v| be the number of examples in *u* and *v* respectively. Also, let $u_{i,j}$ represent the value of $a_i$ in the positive example *i*, and $v_{i,j}$ represent the value of $a_j$ in the negative example *i*.

The algorithm starts by evenly distributing the values in *u* into *m*×*n* partitions. Each partition is labelled as $g_{i,j}$ where $0 \le i < m$ and $0 \le j < n$. Each $g_{i,j}$ consists of $u_{s,t}$, where $i \times \frac{|u|}{m} \le s < (i+1) \times \frac{|u|}{m}$ and $j \times \frac{k}{n} \le t < (j+1) \times \frac{k}{n}$.

Similarly *v* is partitioned in to *n* partitions. Each partition is labelled as $h_i$, where $0 \le i < n$. Each $h_i$, consists of $v_{j,i}$, for $i \times \frac{k}{n} \le t < (i+1) \times \frac{k}{n}$ and for $0 \le j < |v|$.

The algorithm then transfers the values in $g_{i,j}$ and $h_j$ to $p_{i,j}$. Then each $p_{r,c}$ will compute:

$$w_{i,j}^{s} = u_{i,j} \otimes v_{s,j} \qquad (1)$$

Where $r \times \frac{|u|}{m} \le i < (r+1) \times \frac{|u|}{m}$, $c \times \frac{k}{n} \le j < (c+1) \times \frac{k}{n}$, and $0 \le s < |v|$. $u_{i,j} \otimes v_{s,j} = u_{i,j}$ if $u_{i,j} = v_{s,j}$, and $u_{i,j} \otimes v_{s,j} = null$, if $u_{i,j} \ne v_{s,j}$.

At last, each $p_{r,c}$ will compute:

$$Z_{i,j} = w_{i,j}^{0} \cap w_{i,j}^{1} \cap w_{i,j}^{2} \cap ... \cap w_{i,j}^{|v|-1} \qquad (2)$$

Where $r \times \frac{|u|}{m} \le i < (r+1) \times \frac{|u|}{m}$ and $c \times \frac{k}{n} \le j < (c+1) \times \frac{k}{n}$.

## 5 Cost Analysis

The cost of the proposed parallel algorithm consists of communication cost and computation cost, which we will denote as $C_{comm}$ and $C_{comp}$ respectively [**9**]. The communication cost is the time taken to transfer the *g* and the *h* values from processor $p_{0,0}$ to processor $p_{m-1,n-1}$, while the computation cost is the time taken to compute the *w* and *z* values by any one of the processors, say $p_{m-1,n-1}$.

The communication cost is further divided into $C_{startup}$ cost and $C_{data}$ cost. The $C_{startup}$ is the cost of packing the data to be transferred at the source and then unpacking it at the destination. The $C_{data}$ is the cost of transferring one attribute value from one processor to another.

The communication cost of the proposed parallel algorithm is the cost of transferring the *g* and the *h* values from $p_{0,0}$ to $p_{m-1,n-1}$. The cost of transferring the *g* values is $(t_{startup} + \frac{|u|}{m} \times \frac{k}{n} \times t_{data}) \times (m+n-2)$ and the cost of transferring the *h* values is $(t_{startup} + |v| \times \frac{k}{n} \times t_{data}) \times (m+n-2)$. Therefore, the total communication cost is

$$C_{comm} = (2 \times t_{startup} + \frac{k}{n} \times t_{data} \times (\frac{|u|}{m} + |v|)) \times (m+n-2) \qquad (3)$$

Computation cost of the proposed parallel algorithm is the cost of computing the *w* and the *z* values. The cost of computing *w* values is $\frac{|u| \times k}{m \times n} \times |v|$ The cost of computing the *z* values is $\frac{|u| \times k}{m \times n} \times (|v| - 1)$. Therefore, the total computation cost is

$$C_{comp} = \frac{|u| \times k \times (2 \times |v| - 1)}{m \times n} \qquad (4)$$

Hence, the total cost of the proposed parallel algorithm is:

$$C_p = C_{comp} + C_{comm} \qquad (5)$$

**Computation/Communication Ratio:** The communication complexity is $O(\frac{k}{m} \times |u|)$ and the computation complexity is $O(\frac{k}{m} \times \frac{|v|}{n} \times |u|)$.

The ratio of $\frac{t_{comp}}{t_{comm}}$ shows that as $|U|$ increases, the effect of the communication cost decreases, thus the performance of the parallel algorithms increases.

**Speedup:** Speed up is the measure of how much faster the parallel algorithm is compared to the fastest single processor algorithm known to solve the same problem. The cost of the single processor algorithm discussed in Section 3.1 is $C_s = |u| \times |k| \times (2 \times |v| - 1)$. Hence, for a given $m \times n$ mesh, the speedup, which is $\frac{C_s}{C_p}$ will be closer to $m{\times}n$ as $|U|$ increases.

## 6 Conclusion

In short, through this paper, on the basis of studying the limitations of the basic rough set model, we present an extended rough set model that is based on a family set of tolerance relations between objects when given a set of tolerance relations. This model inherits most of the characteristics of the basic model of rough set; and it also has a better effect of approximation classification. And the concepts of general and specific tolerance information systems are presented, so they can be further used to illustrate the relationships between objects in various of real applications.

In the extended rough set model based on the tolerance relations proposed here, according to normal primitive data tables, we could set up a corresponding tolerance information system. On the tolerance information system, we studied the uses of the negative and positive examples concepts from Extension Matrix; from there we proposed an algorithm that gave us one near-optimal attribute reduct based on both extension and relation matrices (Relational Extension Matrix) that is memory saving. Based on this algorithm we further developed a parallel version. Due to the nature of the relational extension matrices used in our algorithm 3.1, the parallel version is very efficient.

**Acknowledgements:** This work has been supported by King Fahd University of Petroleum and Minerals.

## References

[1] Pawlak Z. Rough Sets: *Theoretical Aspects of Reasoning about Data. Dordrecht:* Kluwer Acasemic Publishers, 1991.

[2] Pawlak Z, Grzymala-Busse J, Slowinski R, Ziarko W. *Rough sets. Communications of the ACM*, 1995, 38(11): 89-95.

[3] Slowinski R. *Intelligent Decision Support: Handbook of Applications and Advances of Rough Sets Theory*. Dordrecht: Kluwer Academic Publishers, 1992.

[4] Ziarko W P ed. *Rough Sets, and Fuzzy Sets and Knowledge Discovery* (RSKD'93). London: Springer-Verlag, 1994.

[5] 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 Sets Theory.* Dordrecht: Kluwer Academic Publishers, 1992, 331-362.

[6] Skowron A, Stepaniuk J. Generalized approximation spaces. In: *Lin T Y ed. Conference Proceedings of the Third International Workshop on Rough Sets and Soft Computing* (RSSC'94). San Jose, California, USA, 1994, 156-163.

[7] Hong, J. R. AE1: An Extension Matrix Approximate Method for the General Covering Problem. *International Journal of Computer and Information Science* 1985, 14: 421-437.

[8] Hong J. R. and Uhrik C., The Extension Matrix Approach to Attribute-Based Learning, *Progress in Machine Learning*, I. Bratko and N. Lavrac (Eds.), Wilmslow: Sigma Press, England, 1987.

[9] Wilkson B. and Allen M. *Parallel programming: Techniques and applications using networked workstations and parallel computers*. Prentice Hall, 2005.

[10] Zhou P. L. *Data Mining Using an Extending Rough Set Model*. Monash University, Australia, 2001.

[11] Zhou P. L. Dynamic Reducts of Tolerance Information Systems. The *International Conference on Artificial Intelligence* (ICAI'03). Las Vegas, Nevada, USA, 2003.