Spring 2014

Attribute Selection Methods in Rough Set Theory

Xiaohan Li
San Jose State University

Attribute Selection Methods in Rough Set Theory

A Writing Project

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Xiaohan Li

May 2014

The Designated Project Committee Approves the Project Titled

Attribute Selection Methods in Rough Set Theory

by

Xiaohan Li

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSÉ STATE UNIVERSITY

May 2014

Dr. Sami Khuri

Dr. Melody Moh

Dr. Chris Pollett

ABSTRACT

ATTRIBUTE SELECTION METHODS IN ROUGH SET THEORY

by

Xiaohan Li

Attribute selection for rough sets is an NP-hard problem, in which fast heuristic algorithms are needed to find reducts. In this project, two reduct methods for rough set were implemented: particle swarm optimization and Johnson's method. Both algorithms were evaluated with five different benchmarks from the KEEL repository. The results obtained from both implementations were compared with results obtained by the ROSETTA software using the same benchmarks. The results show that the implementations achieve better correction rates than ROSETTA.

Table of content

1 Introduction

1.1 Project overview

This project focuses on learning rough set theory and implementing a classifier based on RST. For reduct methods, two algorithms were implemented: Johnson's algorithm and PSO algorithm. The proposed algorithms were evaluated by five different data sets from the KEEL repository, including the SSLD problem. In the evaluation, a ten-fold cross validation was used to minimize the side effects of choosing a training set and a test set. The correction rate of each run was measured as the performance of the algorithm. The results of my classifier (using both Johnson's and PSO) were compared with the results obtained by running ROSETTA (using Johnson's and GA) in terms of the correction rate.

1.2 Rough set theory

The Rough Set Theory (RST) was introduced by Z. Pawlak in 1982 as an extension of the traditional set theory in order to better deal with uncertain and fuzzy knowledge [1]. Since uncertainty and vagueness widely exist in the real world, RST has been found to be a powerful mathematical tool in processing real world data sets. Recently, RST has attracted great attention from researchers, and many applications of RST have been actively proposed, especially in the machine-learning domain [2, 3].

1.3 Feature selection

Feature selection is an important step in RST-based machine learning. While generating a decision rule, it is expected to contain only a subset of attributes (features) that are the most informative. All other attributes are removed from the rule with minimum

information loss [4]. The optimal solution of feature selection is a subset of features, called a reduct, with a minimal number of features. An exhaustive solution is to use boolean reasoning laws to find out all reducts, and then to choose the one with minimal attributes. Obviously, this only works for simple data sets. When the number of attributes becomes larger, the number of possible reducts is always very big. For N attributes, there are 2^N subsets of attributes, and the exhaustive method is an NP-hard problem [5]. In practice, heuristic algorithms have to be considered.

1.4 Reduct algorithms

Johnson's algorithm is a simple greedy reduct algorithm. The algorithm always picks the attribute with the maximum count of appearances in each iteration. Johnson's algorithm does not guarantee finding the optimal reduct [4]. Genetic Algorithm (GA) mimics the behavior of natural evolution. It is a widely used heuristic algorithm for finding reducts. One existing implementation of the GA reduct algorithm is included in the ROSETTA toolkit, which is designed for analyzing tabular data within the RST framework [6]. Particle swarm optimization (PSO) is another heuristic algorithm inspired by the behavior of flying birds or fish schools. PSO finds a solution by using a swarm of particles 'flying' throughout the problem space [7]. Studies show that PSO has strong discoverability and often performs better than GA.

1.5 Seven-segment LED display problem

Seven-segment LED display (SSLD) problem is a classification problem, which comes from the KEEL repository [8]. I introduce the SSLD problem here in order to use it as an

example for introducing RST. A seven-segment light emission diode (LED) display is a form of electronic display devices for displaying decimal numbers. As shown in Figure 1, a digit is composed by seven light-emitting diodes, one for each segment. The letters A to G refer to pins that control the number displayed.

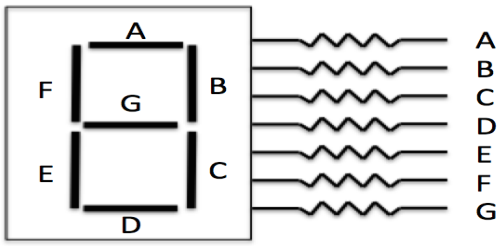


Figure 1: The seven-segment LED display [9]

Table 1 shows the relationship between inputs in pins A – G and the digit displayed in the seven-segment LED display. In this encoding, 1 represents a high electrical level and 0 represents a low electrical level for a pin.

Table 1: Binary encoding for digit 0 - 9 [9]

Digit	A	B	C	D	E	F	G
0	1	1	1	1	1	1	0
1	0	1	1	0	0	0	0
2	1	1	0	1	1	0	1
3	1	1	1	1	0	0	1
4	0	1	1	0	0	1	1
5	1	0	1	1	0	1	1
6	1	0	1	1	1	1	1
7	1	1	1	0	0	0	0
8	1	1	1	1	1	1	1
9	1	1	1	1	0	1	1

The goal of the problem is to predict which digit will be shown in the display for the given inputs in pins A - G. It would be an easy problem if there was no noise introduced. However, consider the case where the input data has a 10% probability of having one of its values inverted. In this case, the simple solution that uses Table 1 to translate encodings into digits will not work. [8]

For example, digit 8 is encoded by 1111111 in Table 1. If the first bit is inverted (shown in Figure 2), the encoding will change to 0111111, and there are no matching digits in Table 1.

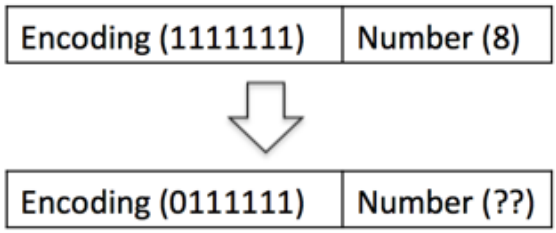


Figure 2: Bit invert example 1

Consider another case shown in Figure 3. Given the encoding 1111011, we cannot decide what the original encoding was before the bits were inverted. The digit might be 9 if no bit has been inverted, or 3, 5 and 8 if one bit has been inverted.

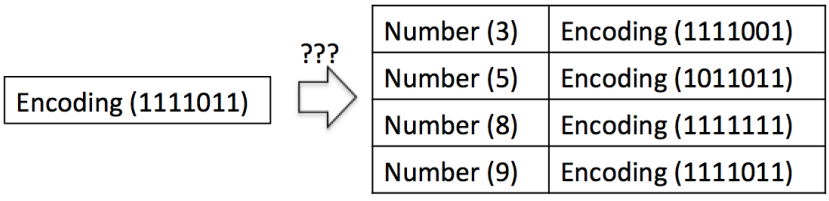


Figure 3: Bit invert example 2

In the KEEL repository, there are 500 SSLD data instances that come from a laboratory. Table 2 shows the format of the data instances.

Table 2: Format of data instances for the SSLD problem [8]

Attribute	Domain
Led1	[0,1]
Led2	[0,1]
Led3	[0,1]
Led4	[0,1]
Led5	[0,1]
Led6	[0,1]
Led7	[0,1]
Number	{0,1,2,3,4,5,6,7,8,9}

The format of the data includes eight attributes. Attributes Led1 – Led7 correspond to A – G in Table 1. The attribute Number corresponds to Digit in Table 1.

The challenges of solving the SSLD problem include:

- It is unknown why the inputs were inverted. It might be caused by unstable voltage, environment temperature, or other reasons.
- The known information about the problem is limited. The KEEL dataset only includes inputs and output, without providing other information.

The SSLD problem is a classification problem in machine learning. Machine learning studies how to automatically learn to make accurate predictions based on observations of

labeled training examples [10]. To solve a classification problem, a machine-learning algorithm is used to generate rules by processing a labeled training set, and then it uses these rules to classify unseen objects into a given set of categories. Figure 4 shows the workflow of solving a classification problem.

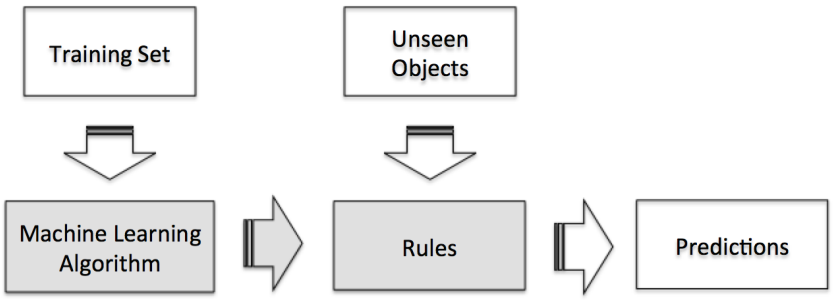


Figure 4: The workflow of solving a classification problem [10]

The rest of this work is divided into the following chapters. Chapter 2 introduces the fundamentals of rough set theory. Chapter 3 discusses the implementation of the rough set classifier. Chapter 4 discusses the principles and implementations of the reduct algorithms. Chapter 5 compares the experimental results and concludes the article.

2 Preliminary of rough set theory

2.1 Information system

An information system is a table with observations (called objects) as rows, features (called attributes) as columns and discrete values as entries. The information system is denoted as $I = (U, A)$ where U is the universe and non-empty set of objects in the table, and A denotes the collection of attributes that are used to describe objects. There are two kinds of attributes: conditional attributes and decisional attributes. Note that $A = C \cup D$. The decisional attributes D determine which class an object belongs to. The conditional attributes C are all other attributes except decisional attributes. [7, 11]

For the SSLD problem, Table 3 is an information system, which includes 8 objects. Led1, Led2, Led3, Led4, Led5, Led6, Led7 and Number are 8 attributes of the information system. Led1 - Led7 are conditional attributes, and Number is a decisional attribute.

Table 3: Information system example

Object	Led1	Led2	Led3	Led4	Led5	Led6	Led7	Number
O1	1	1	1	0	0	1	0	7
O2	1	0	1	1	1	0	0	1
O3	0	0	1	1	0	0	0	3
O4	1	0	1	1	1	1	1	2
O5	1	0	1	1	1	1	1	3
O6	1	0	1	1	1	1	1	8
O7	1	1	1	0	1	1	0	0
O8	1	1	1	0	1	1	0	9

2.2 Indiscernible relation and equivalence class

For any subset of attributes $P \subseteq A$ and subset of objects $X \subseteq U$, the indiscernible relation is defined as [7]:

$$IND(P) = \{(x, y) \in U \mid \forall a \in P, a(x) = a(y)\} \quad (1)$$

If only considering a subset of attributes P but not considering other attributes, two objects might be indiscernible with each other, then we say they are indiscernible. The equivalence class of $IND(P)$ is denoted as $[x]_p$, which means that $\forall y \in [x]_p$, (x, y) are indiscernible to each other. [7]

For the example in Table 3, if only considering conditional attributes (Led1 - Led7), then {O4, O5, O6} is an equivalence class and {O7, O8} is another equivalence class, because O4, O5 and O6 are indiscernible from each other, and O7 and O8 are indiscernible from each other. Table 4 is a full list of equivalence classes.

Table 4: Equivalence class examples

Equivalence class	Encoding (Led1 – Led7)	Number
E1 = {O1}	1110010	{7}
E2 = {O2}	1011100	{1}
E3 = {O3}	0011000	{3}
E4 = {O4, O5, O6}	1011111	{2, 3, 8}
E5 = {O7, O8}	1110110	{0, 9}

2.3 Lower and upper approximation

Based on the definition of an indiscernible relation and an equivalence class, lower and upper approximations are defined as [7]:

$$\underline{P}X = \{x \in U \mid [x]_p \subseteq X\} \tag{2}$$

$$\overline{P}X = \{x \in U \mid [x]_p \cap X \neq \emptyset\} \tag{3}$$

where $\underline{P}X$ denotes the lower approximation and $\overline{P}X$ denotes the upper approximation. Pawlak defines a rough set as a pair of lower approximation and upper approximation [7].

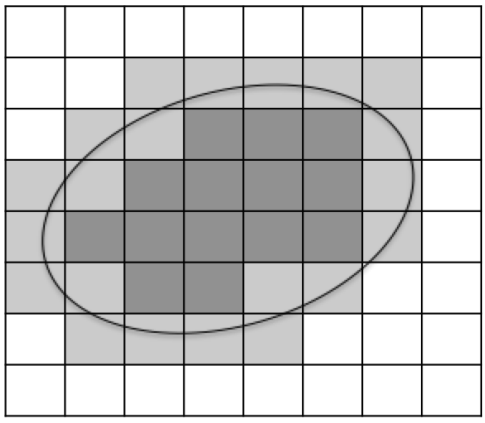


Figure 5: A representation of the rough set [11]

Figure 5 gives a representation of the concept of a rough set. In the figure, the squares denote equivalence classes, and the ellipse denotes the target set X. Since objects belonging to the same equivalence class are indiscernible from each other, equivalence classes are the smallest granularity in the information system. Obviously, based on the equivalence classes (the squares), we cannot exactly define the ellipse. RST solves this problem by defining a pair of approximations, the lower approximation (dark grey) and

the upper approximation (dark and light grey) [11]. The lower approximation includes all equivalence classes that are entirely included by the ellipse, while the upper approximation includes all equivalence classes in the lower approximation and those that are partly inside the ellipse. The figure shows that a rough set contains all the information based on the known attributes.

Decision class 3 in Table 4, for example, can be defined by a lower approximation {E3} and an upper approximation {E3, E4}.

2.4 Discernibility matrix

The discernibility matrix is a |U| × |U|matrix with entries defined as [4]:

$$c_{ij} = \{a \in C \mid a(x_i) \neq a(x_j)\}\, i, j = 1, \ldots, |U| \quad (4)$$

The c_{ij} contains attributes whose values are different between object i and object j. For the equivalence classes in Table 4, one can build the discernibility matrix given in Table 5.

Table 5: Discernibility matrix example

	E1	E2	E3	E4	E5
E1	∅				
E2	Led2, Led4, Led5, Led6	∅			
E3	Led1, Led2, Led4, Led6	Led1, Led5	∅		
E4	Led2, Led4, Led5, Led7	Led6, Led7	Led1, Led5, Led6, Led7	∅	
E5	Led5	Led2, Led4, Led6	Led1, Led2, Led4, Led5, Led6	Led2, Led4, Led7	∅

The discernibility matrix specifies attributes that are discerning different equivalence classes. In the example

- Item (E2, E2) is empty, because equivalence class E2 cannot be discerned from itself.
- Item (E2, E3) is Led1 and Led5 because by comparing equivalence class E2 and E3, only conditional attributes Led1 and Led5 are different.
- Not all entries need to be filled because the matrix is symmetrical.

2.5 Discernibility function

Discernibility function is a boolean function constructed for each equivalence class. This function is true for all attribute combinations that discern this object from other objects with a different decision [11]. A discernibility function f_D of m boolean variables $(a_1^*, \ldots, a_m^*)$, corresponding to attributes $(a_1, \ldots, a_m)$, is defined as [4]:

$$f_D(a_1^*, \ldots, a_m^*) = \wedge \{\vee c_{ij}^* \mid 1 \leq j \leq i \leq |U|, c_{ij} \neq \emptyset\} \text{ where } c_{ij}^* = \{a^* \mid a \in c_{ij}\} \quad (5)$$

For example, according to Table 5, discernibility function f_{E1} that distinguishes E1 from all the other equivalence classes is

f_{E1}= (Led2 ∨ Led4 ∨ Led5 ∨ Led6) ∧ (Led1 ∨ Led2 ∨ Led4 ∨ Led6) ∧ (Led2 ∨ Led4 ∨ Led5 ∨ Led7) ∧ (Led5).

2.6 Reduct

A reduct is the minimal subset of attributes that enables the same discernibility as the whole set of attributes [12]. For a discernibility function, since attributes in a reduct already contain all the information, all other attributes can be removed without losing any

information. One way to produce a reduct is to change the discernibility function from conjunctive normal form to disjunctive normal form [13].

For example:

f_{E1} = (Led2 ∨ Led4 ∨ Led5 ∨ Led6) ∧ (Led1 ∨ Led2 ∨ Led4 ∨ Led6) ∧ (Led2 ∨ Led4 ∨ Led5 ∨ Led7) ∧ (Led5)

= (Led1 ∨ Led2 ∨ Led4 ∨ Led6) ∧ (Led5)

= (Led1 ∧ Led5) ∨ (Led2 ∧ Led5) ∨ (Led4 ∧ Led5) ∨ (Led5 ∧ Led6)

Therefore, the four reducts of f_{E1} are {Led1, Led5}, {Led2, Led5}, {Led4, Led5}, {Led5, Led6}, which are subsets of disjunctive normal form with minimum subsets.

2.7 Decision rules

Consider an information system $I = (U, A)$ and $A = C \cup D$. Every $x \in U$ determines a sequence of $c_1(x), \ldots, c_n(x), d_1(x), \ldots, d_m(x)$ where $\{c_1, \ldots, c_n\} = C$ and $\{d_1, \ldots, d_m\} = D$ is called decision rules induced by x and denoted by $c_1(x), \ldots, c_n(x) \rightarrow d_1(x), \ldots, d_m(x)$. The number $\text{support}_x(C, D) = |C(x) \cap D(x)|$ is called the support of the decisional rule. The support is a widely used quality measure for rules. [14]

In the SSLD example, according to the formula and Table 4, a decision rule induced by equivalence class E1 should be

IF Led1 = 1 and Led2 = 1 and Led3 = 1 and Led4 = 0 and Led5 = 0 and Led6 = 1 and Led7 = 0 THEN Number = 7

We can safely remove attributes that are not included in the reduct because a reduct has

the same discernibility as the original attribute set. For example, if the reduct is {Led1, Led5}, then the decision rule can be changed to

IF Led1 = 1 and Led5 = 0 THEN Number = 7

that contains the same information as the original rule.

The next chapter elaborates on the implementation of the rough set classifier.

3 Implementation of classifier

3.1 Workflow

The overall workflow of using the rough set theory to solve the SSLD problem or other classification problems contains three main steps: data import, rule generation, and classification, as shown in Figure 6. The inputs of the algorithm are a training set and a test set, while the outputs of the algorithm are the predicted classes for the test set.

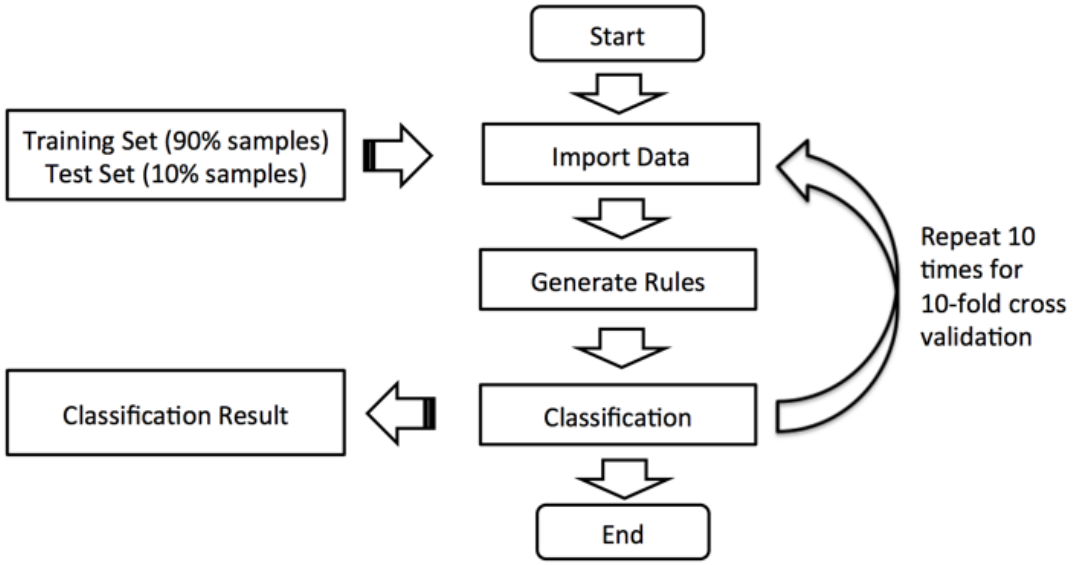


Figure 6: Workflow for the classifier

In Figure 6, the process is repeated ten times because we use the ten-fold cross validation to minimize the side effects of choosing the training set and the test set. For each iteration, 90% of objects in the data set are used as the training set, and 10% are used as the test set. Since the KEEL repository already includes the ten-fold data set, there is no

need to divide it in the implementation.

3.2 Data import

In this step the classifier imports the training set and the test set from a file system into memory, and then generates equivalence classes for the training set. Each data set file in the KEEL repository has the following metadata [8]:

@relation: Name of the data set

@attribute: Description of an attribute (one for each attribute)

@inputs: List with the names of the input attributes

@output: Name of the output attribute

@data: Starting tag of the data

One requirement of the classifier is that the classifier should be able to work on multiple datasets, which means the classifier should not be designed to solve a particular problem. Information provided by the metadata is a key factor to make a general classifier possible. In fact, the classifier can get all the information needed to process the data set by parsing the metadata. For example, the data set files of the SSLD problem have the following metadata:

@relation led7digit

@attribute Led1 real [0.0, 1.0]

@attribute Led2 real [0.0, 1.0]

@attribute Led3 real [0.0, 1.0]

@attribute Led4 real [0.0, 1.0]

@attribute Led5 real [0.0, 1.0]

@attribute Led6 real [0.0, 1.0]

@attribute Led7 real [0.0, 1.0]

@attribute Number {0,1,2,3,4,5,6,7,8,9}

@inputs Led1, Led2, Led3, Led4, Led5, Led6, Led7

@outputs Number

@data

0.000000, 1.000000, 1.000000, 0.000000, 1.000000, 1.000000, 1.000000, 0

1.000000, 1.000000, 1.000000, 0.000000, 0.000000, 1.000000, 1.000000, 0

1.000000, 1.000000, 1.000000, 0.000000, 1.000000, 1.000000, 1.000000, 0

1.000000, 1.000000, 1.000000, 0.000000, 1.000000, 1.000000, 1.000000, 0

0.000000, 0.000000, 1.000000, 0.000000, 0.000000, 1.000000, 0.000000, 1

…

Note that in the example

- The @attribute tag indicating Led1 - Led7 are real values and Number is an integer. The classifier knows how to store and to compare the values of two attributes through the types of attributes.
- The @inputs tag indicates the consequence of the attributes in each row of the data file. In the example Led1, Led2, Led3, Led4, Led5, Led6, Led7, the first value in a row of data is for the attribute Led1, and the second value belongs to the attribute Led2, and so on.
- The @outputs tag indicates that Number is a decisional attribute. Therefore, the classifier knows it should compare the Number when it wants to see if two objects

belong to the same class.

- The @data tag indicates the end of metadata. The classifier knows it is time to read data rows after this tag.
- The rest of the file contains the objects in the data set, in which one object is represented by a row, with a comma separating two consecutive values.

After the data sets are imported, the classifier generates equivalence classes. As discussed before, an equivalence class represents a group of training instances that are indiscernible from their conditional attributes. Equivalence classes are generated by merging training instances having the same conditional attributes.

3.3 Generation of rules

In this step, the classifier first produces discernibility functions for equivalence classes. Second, it simplifies discernibility functions and generates reducts. Finally, the classifier generates IF-THEN rules and measures the support of each decision rule. However, finding a reduct is time consuming for complex data sets. In order to accelerate the processing time, the classifier uses parallel computation to split the workload among multiple CPUs. Technically, this is achieved by using multiple threads. Modern operating systems allocate long running, CPU-intensive threads into different CPUs if multiple CPUs or multiple cores are available in the system. The classifier uses a thread pool to manage threads. A thread pool consists of multiple work threads, and a new task will be assigned to a free thread in the pool. If there is no free thread in the pool, which means the resources are already fully utilized, then new tasks must wait until one thread becomes available. When a thread finishes its work, the thread pool assigns a new task to

it instead of destroying it. This reduces the overhead for destroying and creating threads. The number of threads in the pool is the same as the number of CPUs in the system. Ideally, if the number of threads in the pool is N, then the parallel version runs N times faster than one thread version. On my laptop, the parallel version runs eight threads at the same time. Figure 7 shows that the CPU usage is 789.8%, which means the 8 cores in the system are almost fully utilized.

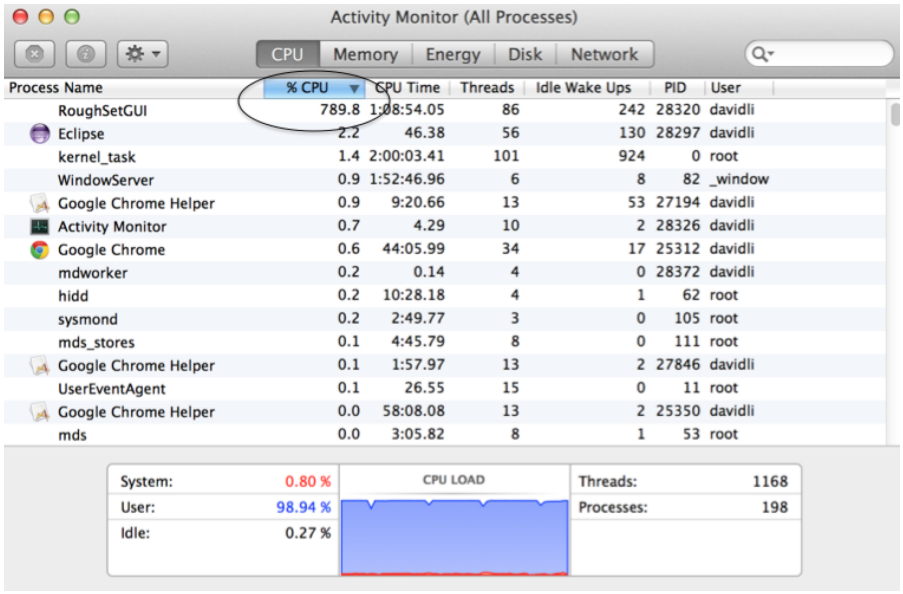


Figure 7: The CPU usage when parallelization is enabled

Figure 8 shows the CPU usage where the parallelization setting was disabled. For the same task, the CPU usage is 104.6% in Figure 8, which means that only one core is working, and all other cores are in the idle state.

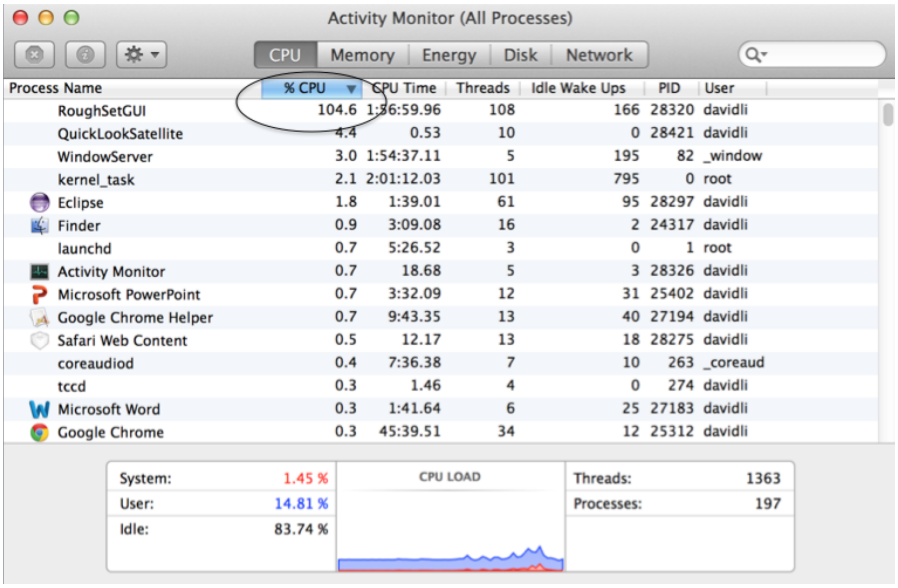


Figure 8: The CPU usage when parallelization is disabled

3.4 Classification

Classification is the process of reading conditional attributes and predicting the value of decisional attributes by using rules. In this step, the test set is used to measure the classification performance of our algorithms. At first we assume the test object's decision class is unknown. The test object is then compared to the IF-part of each rule. If it is matched, the THEN-part of the rule votes what decision class the object belongs to. The class of the object is assigned based on the highest number of votes, also known as the majority vote.

For example, consider we have three rules as shown in Table 6 and a test object shown in Table 7.

Table 6: Rule examples

Name	Rule	Support
R1	IF Led1 (1) AND Led2 (1) AND Led3 (1) AND Led4 (0) AND Led5 (1) THEN Number (0) OR Number (9)	15, 3
R2	IF Led1 (1) AND Led2 (1) AND Led3 (1) AND Led4 (0) AND Led7 (0) THEN Number (7)	2
R3	IF Led1 (1) AND Led2 (1) AND Led3 (1) AND Led5 (1) AND Led7 (0) THEN Number (8)	2

Table 7: Test object example

Conditional Attributes (Led1 - Led7)	Number
1110110	Unknown

This test object matches with all three rules (R1-R3) in Table 6. R1 casts 15 votes to the number 0 and casts 3 votes to the number 9. R2 casts 2 votes to the number 7. R3 casts 2 votes to the number 8.

Table 8: Number of votes

Number	Votes
0	15
7	2
8	2
9	3

Since the number 0 receives the highest votes (shown in Table 8), the classifier predicts the output is the number 0 for this test object.

The ten-fold cross validation is used to better utilize observations. Figure 9 is a graphical representation of the ten-fold cross validation, where each column indicates one iteration of training and classification. In the ten-fold method, the data set is randomly divided into ten subsets (already done by the KEEL repository). In the first iteration, the first subset is used as the test set, and the remaining nine are used to derive the rules. Then, in the second iteration, subset number 2 is used as the test set and so on. This process is repeated ten times.

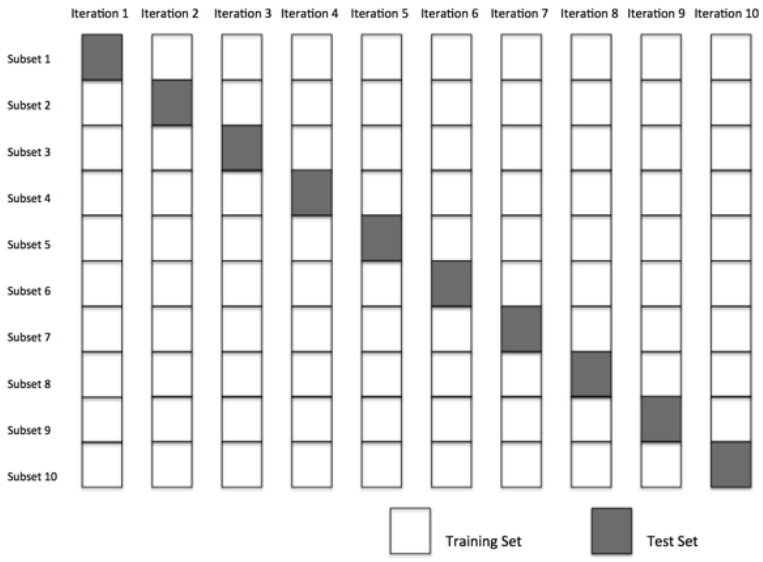


Figure 9: Ten-fold cross validation [11]

The performance of the algorithm is measured using correction rates of classification results. For each output, the correction rate is the fraction of instances correctly classified

in all ten folds.

3.5 Graphical user interface

In this project, the rough set classifier is implemented using the Java programming language. The graphical user interface (GUI) is shown in Figure 10.

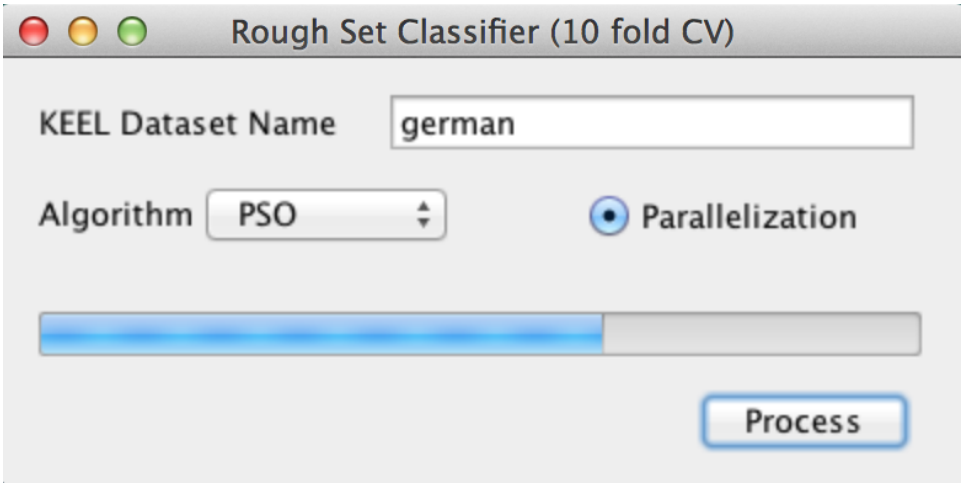


Figure 10: The GUI of the classifier

The function of each control in the GUI is

- KEEL Dataset Name: Input the name of the data set to process. Five data sets have already been integrated in the project, and they are ‘led7digit’, ‘tic-tac-toe’, ‘breast,’ ‘german’ and ‘zoo’.
- Algorithm: Select the algorithm. Available algorithms are Johnson and PSO.
- Parallelization: Set the parallelization setting on or off.
- Progress bar: Show the progress for processing each fold.
- Process: Click the button to start processing the data set.

The outputs will be shown in a separate command line window, or a console tab in Eclipse.

The next chapter begins by introducing Johnson's algorithm and the PSO algorithm, and then it discusses how to use these algorithms to solve the reduct problem.

4 Reduct algorithms

4.1 Johnson's algorithm

Johnson's algorithm is a heuristic algorithm using a greedy technique. The idea of Johnson's algorithm is that it always selects the attribute most frequently occurring in the clause. The algorithm is described as follows:

```
JOHNSON (V, f)
{
        V: set of attributes
        f: a discernibility function
        S ← ∅; //current reduct candidate
        while (f ≠ ∅)
        {
                bCount = 0; //max count so far
                for each( x ∈ V )
                {
                        /*
                        * count(x, f) returns number of x in f
                        */
                        c = count (x, f);
                        if (c > bCount)
                        {
                                bCount = c;
                                bAttr ← x;
                        }
                }
                S ← S ∪ bAttr;
                /*
                * removeAttr(f, bAttr) removes all clauses in f containing bAttr
                */
                f ← removeAttr(f, bAttr);
        }
        return S;
}
```

Johnson's algorithm starts by setting S, the current reduct candidate, to an empty set. Then, the algorithm counts the appearance of each attribute within the clauses. The attribute with the highest count is added into S, and all clauses in f containing this attribute are removed from the discernibility function. The algorithm continues until all clauses are removed from the discernibility function, the algorithm then returns S as a reduct. [4]

For example, the steps of getting the reduct of f = (Led2 ∨ Led4 ∨ Led6) ∧ (Led1 ∨ Led2 ∨ Led4 ∨ Led6) ∧ (Led2 ∨ Led5 ∨ Led7) ∧ (Led5) are:

1. Count the appearance of the attributes, Led1 = 1, Led2 = 3, Led4 = 2, Led5 = 2, Led6 = 2, Led7 = 1.
2. Led2 is the most frequently occurring attribute, so it adds Led2 into S. The classifier removes all clauses containing Led2 from f. After this step, f = Led5 and S = {Led2}.
3. Count the appearance of attributes in f and it finds that Led5 is the most frequently occurring attribute. Then, remove all clauses containing Led5. After this step, f becomes ∅ and S = {Led2, Led5}.
4. The algorithm finishes because f = ∅, and we get reduct Led2 ∧ Led5.

In Johnson's algorithm, the attribute that appears more frequently is considered to be the most significant. Even though this is not true in all cases, Johnson's algorithm generally finds out a solution close to the optimal [4].

4.2 Particle swarm optimization

Particle Swarm Optimization (PSO) is a creative heuristic searching algorithm developed in 1995 by Kennedy and Eberhart [15]. It is inspired by the behavior of flying birds or fish schools. In PSO, a swarm of particles 'fly' throughout the problem space to find the best solution [7]. A particle has a position and a velocity, which are randomly assigned when it is created. Each particle keeps track of the best location it has been in so far, denoted as pbest. All particles also share a value named gbest, meaning the global best value. For each iteration, particles adjust their velocities according to formula 6 [16]:

$$v_{id} = w \cdot v_{id} + c_1 \cdot rand1 \cdot (pbest_{id} - x_{id}) + c_2 \cdot rand2 \cdot (gbest_d - x_{id}) \quad (6)$$

where

- $d = 1, 2 \ldots N$. N equals the dimension of searching space
- v_{id} is the particle's velocity in dimension d
- x_{id} is the particle's position for dimension d
- $pbest_{id}$ is the particle's individual best position in dimension d
- $gbest_d$ is the global best in dimension d
- w is inertia weight
- c_1 and c_2 are acceleration factors
- rand1 and rand2 are two random numbers with range [0, 1]

The particle's new location x_{id} is defined as given by equation [16]:

$$x_{id} = x_{id} + v_{id} \quad (7)$$

There are three parts in formula 6, which determine the new velocity of a particle. These three parts correspond to inertia, individual thinking and group communication in the real

world [17]. The w, c_1 and c_2 are three constant coefficients for these three parts. A maximum velocity V_{max} defines the maximum velocity allowed. If the new velocity is faster than V_{max}, it should be reduced to V_{max}. To measure the location of a particle, a predefined fitness function is used to calculate how good the position is. The pseudo code of PSO algorithm is as follows [17]:

```
PSO (S, N)
{
        S: the size of swarm
        N: iteration number

        gbest = 0, iter = 0;
        gbestPos; //position of gbest
        /*
        generate particles, randomly set their positions and velocities
        */
        particles ← generateParticles (S) ;
        while (iter < N)
        {
                for each (part in particles)
                {
                        /*
                        *  calculate the fitness function for part
                        */
                        fitness = calculateFitness(part);
                        if (fitness > part.best)
                        {
                                part.best = fitness;
                                part.bestPos ← part.x;
                        }
                        if (fitness > gbest)
                        {
                                gbest = fitness;
                                gbestPos ←  part.x;
                        }
```

```
                    /*
                    * update particles
                    * w, c1, c2 are constants
                    * rand1() and rand2() are two random generator in the range [0, 1]
                    */
                    part.v ← w · part.v + c1 · rand1() · ( part.best - part.x) + c2 ·
                    rand2() · (gbest - part.x);
                    if (part.v > Vmax)
                    {
                            part.v ← Vmax;
                    }
                    part.x ← part.x + part.v;
            }
            iter = iter + 1;
        }

        return gbestPos;
}
```

4.3 Solving reduct problem using PSO

In order to use the PSO to solve reduct problems, we should map concepts of the reduct to corresponding concepts in PSO. These concepts include:

- Encoding of position
- Representation of velocity
- Position update
- Selection of constants
- Fitness function

In the implementation, the position of particles is represented as a binary array. In the binary array, the value 1 means the corresponding attribute is selected, and 0 means the

attribute is not selected [17]. PSO was originally designed to work within continuous searching space. In order to solve discrete problems, Kennedy and Eberhart [18] introduced a binary PSO version. In binary PSO, the velocity of a particle is a float number calculated using formula 6. Note that for a binary array, $pbest_{id}$, $gbest_{id}$ and x_{id} are either 0 or 1, and no other values are allowed. The position is updated according to formula 7 and formula 8.

$$\text{If rand}() < \text{sig}(v_{id}) \text{ then } x_{id} = 1, \text{else } x_{id} = 0 \tag{8}$$

where rand() generates a random number in the range [0, 1], and sig is a sigmoid function, which is defined by formula 9 [7]:

$$\text{sig}(v_{id}) = \frac{1}{1 + e^{-v_{id}}} \tag{9}$$

In my implementation, swarm size is set to 20, max iteration is set to 50, w is set to 0.7298, $c_1 = c_2 = 1.49618$ and $V_{max} = 6.0$. See [7, 18]. The fitness function is defined by formulas 10, 11 and 12:

$$\text{fitness} = \alpha \cdot f_a + \beta \cdot f_b \tag{10}$$

$$f_a = \frac{|T|}{|M|} \tag{11}$$

$$f_b = \frac{|C| - |R|}{|C|} \tag{12}$$

where

- α and β are significances of two factors, $\alpha + \beta = 1$
- $|R|$ is the number of 1 in the position.

- |C| is the total number of attributes.
- |T| is the number of clauses that are covered.
- |M| is the total number of clauses in discernibility function

A reduct should have two features: 1) it should have the same discernibility as the whole set of attributes; 2) it should be the minimal subset of attributes. The fitness function also includes two factors f_a and f_b, which correspond to these two features. α and β define the significance of these two factors. In theory, α is more important than β because we do not want to remove attributes that are essential. But how much more important is α than β? In this work, we performed experiments to find out an appropriate pair of α and β values. The experiment contains a series of tests, in which α was set to 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0 respectively. Five datasets from KEEL were processed to test which one has the best correction rates. The experiment result is shown in Figure 11.

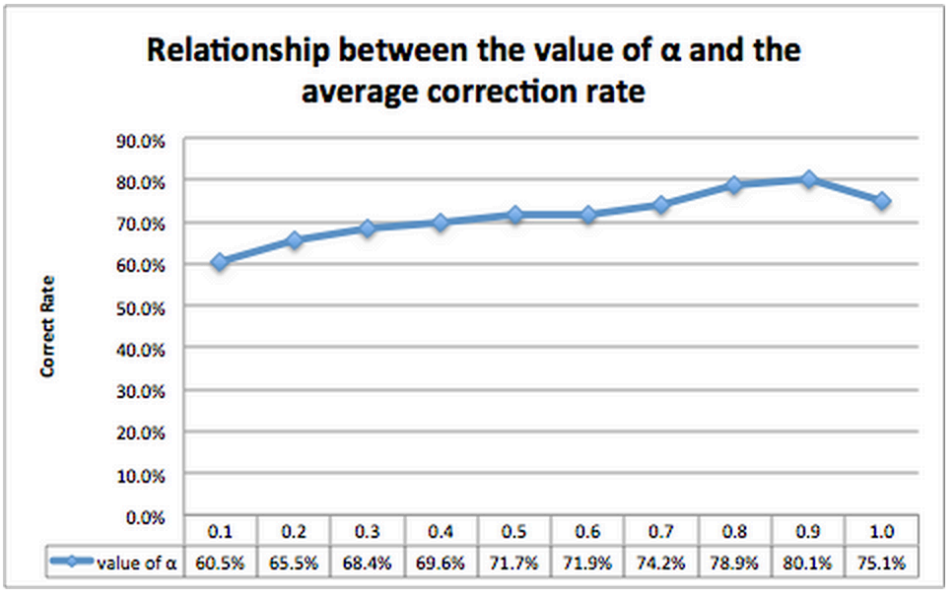


Figure 11: Average correction rates for different α

Figure 11 shows that, on average, the correction rate increases according to the increase of α up to 0.9. After 0.9, the correction rate drops. The experiment shows that $\alpha = 0.9$ is an appropriate value.

In the next chapter, we evaluate the effectiveness of these two reduct algorithms with benchmarks from the KEEL repository. And we compare my results with results obtained by the ROSETTA software using the same benchmarks.

5 Experimental result and discussion

Two reduct algorithms, PSO and Johnson's algorithm, are implemented and tested using five data sets from the KEEL repository. The test results are compared in terms of the correction rates of ten-fold cross validation results. The same data sets have also been processed by using GA and Johnson's algorithm included in ROSETTA toolkit so that we can compare my results with the results of the existing package.

5.1 The led7digit data set

The aim of this data set is to determine what digit is displayed at the 7-segment display according to inputs. Inputs are 7 boolean attributes, one for each segment. If no noise is introduced, the problem is easy. In this case, the input data has a 10% probability of having one of its values inverted. Figure 12 shows the general information about the led7digit data set. [8]

LED Display Domain data set			
Type	Classification	Origin	Laboratory
Features	7	(Real / Integer / Nominal)	(7 / 0 / 0)
Instances	500	Classes	10
Missing values?			No

Figure 12: General information about the led7digit data set [8]

Table 9 shows the ten-fold cross validation results for the four reduct algorithms. The results are shown as the classification correction rates of the classifier using the reduct.

Table 9: Correction rates for the led7digit data set

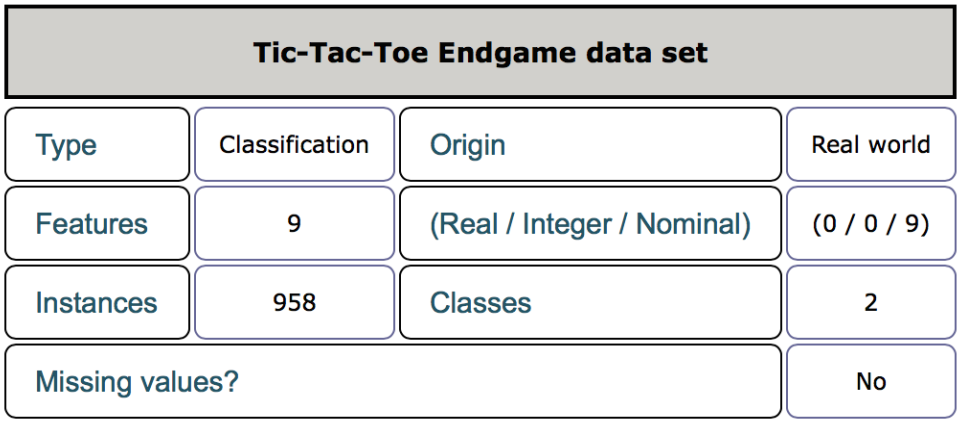
FOLD	Johnson's	PSO	GA (ROSETTA)	Johnson's (ROSETTA)
1	66.0%	68.0%	68.0%	68.0%
2	68.0%	66.0%	66.0%	66.0%
3	74.0%	74.0%	74.0%	74.0%
4	68.0%	70.0%	66.0%	68.0%
5	72.0%	72.0%	74.0%	72.0%
6	76.0%	74.0%	68.0%	68.0%
7	64.0%	64.0%	64.0%	64.0%
8	74.0%	74.0%	72.0%	74.0%
9	68.0%	68.0%	68.0%	68.0%
10	66.0%	66.0%	66.0%	66.0%
Average	69.6%	69.6%	68.6%	68.8%

As can be seen from Table 9, my Johnson's algorithm and PSO algorithm got highest correction rate (69.6%). Johnson's algorithm in ROSETTA got 68.8%, which is slightly better than GA (68.6%).

5.2 The tic-tac-toe dataset

This data set includes possible board configurations at the end of a tic-tac-toe game. The target is to predict if player x will win the game. The general information about the tic-

tac-toe data set is shown in Figure 13. [8]



Tic-Tac-Toe Endgame data set			
Type	Classification	Origin	Real world
Features	9	(Real / Integer / Nominal)	(0 / 0 / 9)
Instances	958	Classes	2
Missing values?			No

Figure 13: General information about the tic-tac-toe data set [8]

Table 10 shows the test results for the tic-tac-toe data set.

Table 10: Test results for the tic-tac-toe data set

FOLD	Johnson's	PSO	GA (ROSETTA)	Johnson's (ROSETTA)
1	87.4%	96.8%	73.7%	77.9%
2	88.4%	100.0%	60.0%	50.5%
3	88.5%	100.0%	85.4%	82.3%
4	86.5%	93.8%	56.3%	57.3%
5	85.4%	100.0%	65.6%	58.3%
6	96.9%	100.0%	64.6%	65.6%
7	86.5%	100.0%	61.5%	59.4%
8	88.5%	99.0%	66.7%	64.6%
9	79.2%	95.8%	67.7%	68.8%
10	89.6%	100.0%	90.6%	91.7%
Average	87.7%	98.5%	69.2%	67.6%

The results show that PSO got the highest average correction rate (98.5%), which is much better than my Johnson's algorithm (87.7%). GA got 69.2%, which is also better than Johnson's algorithm in ROSETTA (67.6%).

5.3 The breast dataset

This data set is about breast cancer. The objects include 9 conditional attributes, and some are linear and some are nominal. The data set includes 201 objects of one class and 85 objects of another class. The general information about the breast data set is shown in Figure 14. [8]

Breast Cancer data set			
Type	Classification	Origin	Real world
Features	9	(Real / Integer / Nominal)	(0 / 0 / 9)
Classes	2	Missing values?	Yes
Total instances	286	Instances without missing values	277

Figure 14: General information about the breast data set [8]

Table 11 contains the test results for the breast data set. The results in Table 11 show the same pattern as the test results of tic-tac-toe data set. That is: PSO (70.3%) is better than my Johnson's algorithm (68.8%); GA (69.2%) is better than Johnson's algorithm in ROSETTA (67.2%).

Table 11: Test results for the breast data set

FOLD	Johnson's	PSO	GA (ROSETTA)	Johnson's (ROSETTA)
1	75.0%	75.0%	67.9%	67.9%
2	67.9%	71.4%	65.5%	69.0%
3	66.7%	70.4%	57.1%	57.1%
4	56.7%	53.3%	75.0%	71.4%
5	62.1%	69.0%	83.3%	63.3%
6	74.1%	81.5%	56.7%	66.7%
7	66.7%	74.1%	82.1%	71.4%
8	75.0%	64.3%	71.4%	64.3%
9	73.1%	73.1%	65.5%	62.1%
10	70.4%	70.4%	67.9%	78.6%
Average	68.8%	70.3%	69.2%	67.2%

5.4 The german dataset

This data set comes from Statlog German Credit Data about bank customers and their bank accounts. The aim is to classify the customer as good or bad. The general information about the german data set is shown in Figure 15. [8]

German Credit data set			
Type	Classification	Origin	Real world
Features	20	(Real / Integer / Nominal)	(0 / 7 / 13)
Instances	1000	Classes	2
Missing values?			No

Figure 15: General information about the german data set [8]

Table 12 contains the test results for the german data set.

Table 12: Test results for the german data set

FOLD	Johnson's	PSO	GA (ROSETTA)	Johnson's (ROSETTA)
1	65.0%	71.0%	67.0%	65.0%
2	70.0%	70.0%	69.0%	70.0%
3	69.0%	69.0%	70.0%	69.0%
4	68.0%	66.0%	68.0%	68.0%
5	73.0%	74.0%	72.0%	74.0%
6	68.0%	66.0%	69.0%	68.0%
7	67.0%	67.0%	70.0%	66.0%
8	69.0%	69.0%	69.0%	70.0%
9	66.0%	73.0%	70.0%	65.0%
10	69.0%	67.0%	68.0%	71.0%
Average	68.4%	69.2%	69.2%	68.6%

As shown in Table 12, PSO and GA got the highest correction rate (69.2%). My Johnson's algorithm got 68.4%, and Johnson's algorithm in ROSETTA got 68.6%.

5.5 The zoo dataset

This is a simple data set used to classify animals in seven predefined classes (mammals, birds and so on). Most of the attributes are boolean values. The general information about the zoo data set is shown in Figure 16. [8]

Zoo data set			
Type	Classification	Origin	Laboratory
Features	16	(Real / Integer / Nominal)	(0 / 0 / 16)
Instances	101	Classes	7
Missing values?			No

Figure 16: General information about the zoo data set [8]

Table 13 contains the test results for the zoo data set. For this data set, Johnson's algorithm in ROSETTA got the highest correction rate (97.7%). My PSO got 96.7%, which is better than my Johnson's (95.7%). GA got the worst correction rate (95.2%).

Table 13: Test results for the zoo data set

FOLD	Johnson's	PSO	GA (ROSETTA)	Johnson's (ROSETTA)
1	83.3%	100.0%	91.7%	100.0%
2	100.0%	100.0%	100.0%	100.0%
3	100.0%	100.0%	100.0%	100.0%
4	100.0%	100.0%	100.0%	100.0%
5	90.0%	90.0%	90.0%	90.0%
6	100.0%	100.0%	100.0%	100.0%
7	90.0%	90.0%	90.0%	100.0%
8	93.3%	86.7%	80.0%	86.7%
9	100.0%	100.0%	100.0%	100.0%
10	100.0%	100.0%	100.0%	100.0%
Average	95.7%	96.7%	95.2%	97.7%

5.6 Summary for the correction rates

Table 14 is a summary of the test results for five individual data sets, shown as the average correction rates for classifiers using these four algorithms.

Table 14: Summary for the results of five data sets

Data set	Johnson's	PSO	GA (ROSETTA)	Johnson's (ROSETTA)
led7digit instance: 500 attribute: 7 decision class: 10	69.6%	69.6%	68.6%	68.8%
tic-tac-toe instance: 958 attribute: 9 decision class: 2	87.7%	98.5%	69.2%	67.6%
breast instance: 286 attribute: 9 decision class: 2	68.8%	70.3%	69.2%	67.2%
german instance: 1000 attribute: 20 decision class: 2	68.4%	69.2%	69.2%	68.6%
zoo instance: 101 attribute: 16 decision class: 7	95.7%	96.7%	95.2%	97.7%
Average	78.0%	80.9%	74.3%	74.0%

Upon comparing the results obtained by the four algorithms, one can conclude that the differences between them are pretty small. On average, the lowest correction rate is 74.0% and the highest is 80.9%. By comparing the result of my Johnson's algorithm and ROSETTA's Johnson's algorithm, my implementation got better results for three data sets and got worse results for two data sets. On average, the correction rate of my Johnson's implementation is 78.0%, which is better than ROSETTA's Johnson's algorithm (74.0%). Note that other factors in the rough set framework might also affect

the correction rate, and a higher correction rate does not mean my Johnson's algorithm is better than ROSETTA's. But this at least shows my implementation of the rough set framework and Johnson's algorithm are correct and produce reasonable results. By comparing the test results of my PSO and my Johnson's algorithm, PSO always got equal or better results. On average, PSO got 80.9% in correction rate, while my Johnson's algorithm got 78.0%. By comparing ROSETTA's GA and ROSETTA's Johnson's algorithm, GA got 74.3% on average correction rate, which is slightly better than ROSETTA's Johnson's algorithm (74.0%). Regarding the results of GA and PSO, PSO is equal or better than GA in all data sets.

5.7 Speed comparison

In this part, I compare the running time of my PSO and my Johnson's algorithm. I do not measure algorithms in ROSETTA because there is no easy way to calculate the running time for ROSETTA. The results are tested on a MacBook Pro with 2.3 GHz Intel Core i7 CPU and 8GB memory. The operating system is OS X 10.9.2. The running time is shown in Table 15.

Table 15: Running time comparison, Johnson's vs PSO

Algorithm	led7digit (ms)	tic-tac-toe (ms)	breast (ms)	german (ms)	zoo (ms)
Johnson's	121	1509	178	2987	50
PSO	2426	67081	7673	107033	2127
Ratio	20.05	44.45	43.11	35.83	42.54

The result shows that Johnson's algorithm is much better with respect to processing time. For these five data sets, Johnson's algorithm is about 20 - 45 times faster than PSO. The time complexity of Johnson's algorithm is $O(N^2M^2)$ where N is the number of attributes, and M is the number of objects in the training set. PSO is $O(SI \cdot N^2M^2)$ where S is the number of particles, and I is the number of iterations. As S and I are constants, the time complexity of two algorithms are all $O(N^2M^2)$. PSO is slower than Johnson's algorithm because Johnson's algorithm has a much smaller constant factor. From the test results, we can find that the ratios do not change as much as the complexities of data sets vary. This supports my conclusion that two algorithms are in the same order.

5.8 Parallelization performance comparison

As previously discussed, parallel computing can improve performance. This section compares the difference in performance between parallelization enabled and disabled. Table 16 shows the different processing times of my PSO classifier when parallelization is enabled or disabled. The results were tested on a MacBook Pro with 2.3 GHz Intel Core i7 CPU and 8GB memory. The operating system is OS X 10.9.2. The different running times are shown in Table 16.

Table 16: Parallelization performance for PSO

Parallelization	led7digit (ms)	tic-tac-toe (ms)	breast (ms)	german (ms)	zoo (ms)
Disabled	2426	67081	7673	107033	2127
Enable	651	15936	1665	25485	471
Ratio	3.73	4.21	4.61	4.20	4.52

The CPU of the test machine includes 4 cores, and each core can run two threads at the same time. So in theory, it can run 8 threads at the same time. The test results show it is about 3 times faster after the parallelization option is enabled. Since parallelization involves some overheads and part of the work does not run in parallel, this seems to be a reasonable result.

Table 17 shows the different processing times of my Johnson's classifier when parallelization is enabled or disabled.

Table 17: Parallelization performance for Johnson's

Parallelization	led7digit (ms)	tic-tac-toe (ms)	breast (ms)	german (ms)	zoo (ms)
Disabled	121	1509	178	2987	50
Enabled	109	1522	170	3015	56
Ratio	0.90	1.01	0.96	1.01	1.12

The results show a different pattern than the previous one. The performances are almost the same whether parallelization is enabled or disabled. In some cases, the performances even become worse when parallelization is enabled. This is expected, because Johnson's reduct algorithm is very fast, and the overhead of parallelization might be bigger than its benefit.

5.9 Conclusion and possible directions for future research

In this work, I used rough set theory to solve the SSLD problem and other classification

problems. The results show that rough set theory is a powerful tool for solving classification problems. Two reduct algorithms, PSO and Johnson's algorithm, are studied, implemented and tested. Among PSO, GA and Johnson's algorithm, Johnson's algorithm is the fastest reduct algorithm. It does not guarantee finding a shortest reduct, but normally finds a solution close to the optimal one. Johnson's algorithm guarantees the discernibility, which means the rules generated from the reducts keep all information in the training set. The test results show that Johnson's correction rate is slightly lower than GA and PSO but not by much. The disadvantage of Johnson's algorithm is that there is no easy way to improve the output quality, because it does not have any parameter to adjust. Compared to Johnson's algorithm, PSO is slower and does not guarantee the discernibility. Instead, my implementation of PSO introduces two factors, α and β, to balance the output's discernibility and length. Since different problems have different emphases on these two factors, PSO can adapt to different problems by changing α and β. For example, the classifier uses a lower α in image recognition problems with acceptable detail loss. PSO is also easily improved by increasing swarm size and iterations. Obviously, this will increase the running time at the same time. These kinds of adjustments make PSO quite a flexible algorithm. Our tests show PSO works better than GA and Johnson's algorithm, and this result is achieved by a relatively small swarm size (20) and iterations (50). A possible future research direction could be to optimize the PSO algorithm by adjusting parameters such as α, β, c_1, c_2, w, swarm size and iteration count.

6 REFERENCES

[1] Pawlak, Zdzisław. "Rough sets." *International Journal of Computer & Information Sciences* 11.5 (1982): 341-356.

[2] Jensen, Richard, and Qiang Shen. "Semantics-preserving dimensionality reduction: rough and fuzzy-rough-based approaches." *Knowledge and Data Engineering, IEEE Transactions on* 16.12 (2004): 1457-1471.

[3] Hongyuan, Shen, Yang Shuren, and Liu Jianxun. "An attribute reduction of rough set based on PSO." *Lecture Notes in Computer Science* 6401 (2010): 695-702.

[4] Jensen, Richard, and Qiang Shen. "Rough set based feature selection: A review." *Rough Computing* (2007).

[5] Zhong, Ning, Juzhen Dong, and Setsuo Ohsuga. "Using rough sets with heuristics for feature selection." *Journal of Intelligent Information Systems* 16.3 (2001): 199-214.

[6] Øhrn, J. Komorowski. "ROSETTA: A Rough Set Toolkit for Analysis of Data, Proc." *Third International Joint Conference on Information Sciences, Fifth International Workshop on Rough Sets and Soft Computing* (1997): March 1-5, Vol. 3, pp. 403-407.

[7] Cervante, Liam et al. "Binary particle swarm optimisation and rough set theory for dimension reduction in classification." *Evolutionary Computation (CEC), 2013 IEEE Congress on* 20 Jun. 2013: 2428-2435.

[8] Alcalá-Fdez, Jesús et al. "KEEL Data-Mining Software Tool: Data Set Repository, Integration of Algorithms and Experimental Analysis Framework." *Journal of Multiple-Valued Logic & Soft Computing* 17 (2011).

[9] Tony, R. (2010, Dec. 15). "7-segment display." *7-segment displays tutorial*. Retrieved

Sep. 16, 2013, from http://www.sentex.ca/~mec1995/tutorial/7seg/7seg.html.

[10] Rob, S. "Machine Learning Algorithms for Classification." *Redirection*. Retrieved Mar.10,2014, from http://www.cs.princeton.edu/~schapire/talks/picasso-minicourse.pdf.

[11] Hvidsten, Torgeir R. "A tutorial-based guide to the ROSETTA system: A Rough Set Toolkit for Analysis of Data." 1 (2006).

[12] Yue, Benxian et al. "A new rough set reduct algorithm based on particle swarm optimization." *Bio-inspired Modeling of Cognitive Tasks* (2007): 397-406.

[13] Jensen, Richard, Qiang Shen, and Andrew Tuson. "Finding rough set reducts with SAT." *Rough Sets, Fuzzy Sets, Data Mining, and Granular Computing* (2005): 194-203.

[14] Pawlak, Zdzisław. "Elementary rough set granules: toward a rough set processor." *Rough-Neural Computing* (2004): 5-13.

[15] Eberhart, Russ C, and James Kennedy. "A new optimizer using particle swarm theory." *Proceedings of the sixth international symposium on micro machine and human science* 4 Oct. 1995: 39-43.

[16] Eberhart, Russell C, and Yuhui Shi. "Particle swarm optimization: developments, applications and resources." *Evolutionary Computation, 2001. Proceedings of the 2001 Congress on* 2001: 81-86.

[17] Wang, Xiangyang et al. "Feature selection based on rough sets and particle swarm optimization." *Pattern Recognition Letters* 28.4 (2007): 459-471.

[18] Kennedy, James, and Russell C Eberhart. "A discrete binary version of the particle swarm algorithm." *Systems, Man, and Cybernetics, 1997. Computational Cybernetics and Simulation., 1997 IEEE International Conference on* 12 Oct. 1997: 4104-4108.