Tabu Search for Attribute Reduction in Rough Set Theory

Abdel-Rahman Hedar*, Jue Wang†, and Masao Fukushima

Department of Applied Mathematics and Physics
Graduate School of Informatics, Kyoto University, Kyoto 606-8501, JAPAN
email: {hedar, wjue, fuku}@amp.i.kyoto-u.ac.jp

July 3, 2006

Abstract

Attribute reduction of an information system is a key problem in rough set theory and its applications. Using computational intelligence (CI) tools to solve such problems has recently fascinated many researchers. CI tools are practical and robust for many real-world problems, and they are rapidly developed nowadays. However, some classes of CI tools, like memory-based heuristics, have not been involved in solving information systems and data mining applications like other well-known CI tools of evolutionary computing and neural networks. In this paper, we consider a memory-based heuristic of tabu search to solve the attribute reduction problem in rough set theory. The proposed method, called tabu search attribute reduction (TSAR), shows promising and competitive performance compared with some other CI tools in terms of solution qualities. Moreover, TSAR shows a superior performance in saving the computational costs.
Keywords: *Computational Intelligence, Granular Computing, Attribute Reduction, Rough Set, Tabu Search.*

1 Introduction

Computational Intelligence (CI) tools and applications have grown rapidly since its inception in the early nineties of the last century [4, 13]. CI tools, which are alternatively called *soft computing*, were firstly limited to fuzzy logic, neural networks and evolutionary computing as well as their hybrid methods [22]. Nowadays, the definition of CI tools has been extended to cover many of other machine learning tools [2, 4, 13]. One of the main CI classes is Granular Computing (GrC) [1, 14], which has recently been developed to cover all tools that mainly invoke computing with fuzzy and rough sets. The theory of rough set (RS) proposed by Pawlak [16] in 1982 emerges as a powerful tool for managing uncertainty that arises from inexact, noisy, or incomplete information. It is shown to be

*Abdel-Rahman Hedar is also affiliated with the Department of Computer Science, Faculty of Computer and Information Sciences, Assiut University, EGYPT (email: hedar@aun.edu.eg).

†Jue Wang is with the Academy of Mathematics and System Science, Chinese Academy of Science, Beijing 710071, P. R. China (email: wjue@amss.ac.cn).

methodologically significant in the domains of artificial intelligence and cognitive science, especially in respect of the representation of and the reasoning with imprecise knowledge, machine learning, and knowledge discovery [16, 17, 18].

Calculation of reducts of an information system is a key problem in RS theory [17, 12, 20]. We need to get reducts of an information system in order to extract rule-like knowledge from an information system. Reduct is a minimal attribute subset of the original data which has the same discernibility power as all of the attributes in the rough set framework. Obviously, reduction is an attribute subset selection process, where the selected attribute subset not only retains the representational power, but also has minimal redundancy. Many researchers have endeavored to develop efficient algorithms to compute useful reduction of information systems, see [14] for instance. Besides mutual information and discernibility matrix based attribute reduction methods, they have developed some efficient reduction algorithms based on CI tools of genetic algorithm, ant colony optimization, simulated annealing, and others [12]. These techniques have been successfully applied to data reduction, text classification and texture analysis [14]. Actually, the problem of attribute reduction (AR) of an information system has made great gain from rapid development of CI tools, see [12] and reference therein.

One class of the promising CI tools is memory-based heuristics, like Tabu Search (TS), which have shown their successful performance in solving many combinatorial search problems [8, 19]. However, the contributions of memory-based heuristics to information systems and data mining applications are still limited compared with other CI tools like evolutionary computing and neural networks. In this paper, we propose a TS-based method, called Tabu Search for Attribute Reduction (TSAR), to solve the problem of attribute reduction of an information system. TSAR uses a 0-1 variable representation of solutions in searching for reducts. A rough set dependency degree function is invoked to measure the solution qualities. The search process in TSAR is a high-level TS with long-term memory. Therefore, TSAR invokes diversification and intensification search schemes besides the TS neighborhood search methodology.

In the literature, much effort has been made to deal with the AR problem, see [3, 10, 11, 12, 20, 21, 23] and references therein. Jensen and Shen [11, 12] have extensively studied CI tools for the AR problem. In their works, three CI methods, GenRSAR, AntRSAR, and SimRSAR, have been presented to solve the AR problem. GenRSAR is a genetic-algorithm-based method and its fitness function takes into account both the size of subset and its evaluated suitability. AntRSAR is an ant-colony-based method in which the number of ants is set to the number of attributes, with each ant starting on a different attribute. Ants construct possible solutions until they reach a rough set reduct. SimRSAR employs a simulated-annealing-based attribute selection mechanism. SimRSAR tries to update solutions, which are attribute subsets, by considering three attributes to be added to the current solution or to be removed from it. Optimizing the objective function attempts to maximize the rough set dependency while minimizing the subset cardinality.

The TSAR method proposed in this paper uses the TS neighborhood search methodology for searching reducts of an information system. TS neighborhood search is based on two main concepts; avoiding return to a recently visited solution, and accepting downhill moves to escape from local maximum information. Some search history information is reserved to help the search process to behave more intelligently. Specifically, the best reducts found so far and the frequency of choosing each attribute are saved to provide the diversification and intensification schemes with more promising solutions. TSAR invokes three diversification and intensification schemes; diverse solution generation, best reduct shaking which attempts to reduce its cardinality, and elite reducts inspiration. The numerical results shown in Section 4 later indicate that the proposed TSAR method is competitive with

some other well-known CI tools for the AR problem in terms of reduct qualities. Moreover, TSAR shows a superior performance in saving the computational costs of the dependency degree function.

The paper is organized as follows. In the next section, we briefly give the principles of rough set and attribute reduction, and tabu search as preliminaries needed throughout the paper. In Section 3, we highlight the main components of TSAR and present the algorithm formally. In Section 4, we report numerical results with TSAR using some well-known datasets. Finally, the conclusion makes up Section 5.

2 Preliminaries

This section highlights the main idea and concepts of rough set and attribute reduction as well as tabu search.

2.1 rough set and Attribute Reduction

An information system is a formal representation of a dataset to be analyzed, and it is defined as a pair $S = (U, \mathbb{A})$, where U is a non-empty set of finite objects, called the universe of discourse, and $\mathbb{A}$ is a non-empty set of attributes. With every attribute $a \in \mathbb{A}$, a set of its values V_a is associated [17]. In practice, we are mostly interested in dealing with a special case of information system called a decision system. It consists of a pair $S = (U, \mathbb{C} \cup \mathbb{D})$, where $\mathbb{C}$ is called a conditional attributes set and $\mathbb{D}$ a decision attributes set.

The RS theory is based on the observation that objects may be indiscernible (indistinguishable) because of limited available information. For a subset of attributes $P \subseteq \mathbb{A}$, the indiscernibility relation is defined by $\mathit{IND}(P)$ [17]:

$$\mathit{IND}(P) = \{(\xi, \eta) \in U \times U | \forall a \in P, a(\xi) = a(\eta)\}.$$

It is easily shown that $\mathit{IND}(P)$ is an equivalence relation on the set U. The relation $\mathit{IND}(P), P \subseteq \mathbb{A}$, constitutes a partition of U, which is denoted $U/\mathit{IND}(P)$. If $(\xi, \eta) \in \mathit{IND}(P)$, then ξ and η are indiscernible by attributes from P. The equivalence classes of the *P*-indiscernibility relation are denoted $[\xi]_P$. For a subset $\Xi \subseteq U$, the *P*-lower approximation of Ξ can be defined as

$$\underline{P}\Xi = \{\xi | [\xi]_P \subseteq \Xi\}.$$

As an illustrative example, Table 1(i) shows a dataset which consists of three conditional attributes $\mathbb{C} = \{a, b, c\}$, one decision attribute $\mathbb{D} = \{d\}$, and six objects $U = \{e1, e2, \ldots, e6\}$. If $P = \{a, b\}$, then objects $e1$, $e2$, and $e3$ are indiscernible, and so are objects $e4$ and $e6$. Thus, $\mathit{IND}(P)$ yields the following partition of U:

$$U/\mathit{IND}(P) = \{\{e1, e2, e3\}, \{e4, e6\}, \{e5\}\}.$$

For example, if $\Xi = \{e1, e4, e5\}$, then $\underline{P}\Xi = \{e5\}$; if $\Xi = \{e2, e3, e6\}$, then $\underline{P}\Xi = \phi$.

Let $\mathit{IND}(P)$ and $\mathit{IND}(Q)$ be indiscernibility relations on U, which are defined by the subset of attributes $P \subseteq \mathbb{A}$ and $Q \subseteq \mathbb{A}$, respectively. An often applied measure is the dependency degree of Q on P, which is defined as follows [17]:

$$\gamma_P(Q) = \frac{|POS_P(Q)|}{|U|},$$

Table 1: An Example of Reducts

(i) A Dataset

U	a	b	c	d
$e1$	0	0	0	1
$e2$	0	0	1	0
$e3$	0	0	2	0
$e4$	1	0	0	1
$e5$	1	1	1	1
$e6$	1	0	2	0

(ii) A Reduced Dataset

U	a	c	d
$e1$	0	0	1
$e2$	0	1	0
$e3$	0	2	0
$e4$	1	0	1
$e5$	1	1	1
$e6$	1	2	0

(iii) A Reduced Dataset

U	b	c	d
$e1$	0	0	1
$e2$	0	1	0
$e3$	0	2	0
$e4$	0	0	1
$e5$	1	1	1
$e6$	0	2	0

where $|F|$ denotes the cardinality of set F and $POS_P(Q) = \bigcup_{\Xi \in U/IND(Q)} \underline{P}\Xi$, called a positive region of the partition $U/\mathit{IND}(Q)$ with respect to P, is the set of all elements of U that can be uniquely classified to blocks of the partition $U/\mathit{IND}(Q)$ by means of P. If $\gamma_P(Q) = 1$, we say that Q depends totally on P, and if $\gamma_P(Q) < 1$, we say that Q depends partially on P. The dependency degree expresses the ratio of all objects of U that can be properly classified to the blocks of the partition $U/\mathit{IND}(Q)$ using the knowledge in P. For example, consider the dataset shown in Table 1(i), and let $P = \{a, b\}$ and $Q = \{d\}$. Then

$$U/\mathit{IND}(Q) = \{\{e1, e4, e5\}, \{e2, e3, e6\}\},$$

$$POS_P(Q) = POS_{\{a,b\}}(\{d\}) = \bigcup\{\{e5\}, \phi\} = \{e5\},$$

$$\gamma_P(Q) = \gamma_{\{a,b\}}(\{d\}) = \frac{|POS_{\{a,b\}}(\{d\})|}{|U|} = \frac{1}{6}.$$

One of the major applications of rough set theory is the attribute reduction, that is, the elimination of attributes considered to be redundant, while avoiding information loss [17, 18]. The reduction of attributes is achieved by comparing equivalence relations generated by sets of attributes. Using the dependency degree as a measure, attributes are removed so that the reduced set provides the same dependency degree as the original. In a decision system, a reduct is formally defined as a subset R of the conditional attribute set $\mathbb{C}$ such that $\gamma_R(\mathbb{D}) = \gamma_{\mathbb{C}}(\mathbb{D})$, where $\mathbb{D}$ is the decision attributes set. A given dataset may have many reducts. Thus the set $\mathfrak{R}$ of all reducts is defined as [17]:

$$\mathfrak{R} = \{R : R \subseteq \mathbb{C}; \gamma_R(\mathbb{D}) = \gamma_{\mathbb{C}}(\mathbb{D})\}.$$

The intersection of all the sets in $\mathfrak{R}$ is called the core,

$$\mathrm{Core}(\mathfrak{R}) = \bigcap_{\mathrm{R} \in \mathfrak{R}} \mathrm{R}.$$

The elements of the core are those attributes that cannot be eliminated without introducing more contradictions to the dataset. In the process of attribute reduction, a set $\mathfrak{R}_{\min} \subseteq \mathfrak{R}$ of reducts with minimum cardinality is searched for:

$$\mathfrak{R}_{\min} = \{R \in \mathfrak{R} : |R| \leq |S|, \forall S \in \mathfrak{R}\}.$$

which is called the minimal reduct set.

Using the example shown in Table 1(i), the dependency degree of $\mathbb{D} = \{d\}$ on all possible subsets of $\mathbb{C}$ can be calculated as:

$$\begin{array}{lll} & \gamma_{\mathbb{C}}(\mathbb{D}) = 1, & \\ \gamma_{\{a,b\}}(\mathbb{D}) = \frac{1}{6}, & \gamma_{\{b,c\}}(\mathbb{D}) = 1, & \gamma_{\{a,c\}}(\mathbb{D}) = 1, \\ \gamma_{\{a\}}(\mathbb{D}) = 0, & \gamma_{\{b\}}(\mathbb{D}) = \frac{1}{6}, & \gamma_{\{c\}}(\mathbb{D}) = \frac{2}{3}. \end{array}$$

So, the minimal reduct set for this example is given by:

$$\mathfrak{R}_{\min} = \{\{a, c\}, \{b, c\}\}.$$

If the minimal reduct $\{a, c\}$ is chosen, then the example dataset shown in Tabel 1(i) can be reduced as in Table 1(ii). On the other hand, Table 1(iii) shows a reduced dataset corresponding to the minimal reduct $\{b, c\}$.

It is well known that finding a minimal reduct is NP-hard [17]. The most primitive solution to locating such a reduct is to simply generate all possible reducts and choose some with minimal cardinality. Obviously, this is an expensive procedure and it is only practical for simple datasets. For most of the applications, only one minimal reduct is required, so all the calculations involved in discovering the rest are pointless. Therefore, an alternative strategy is required for large datasets.

2.2 Tabu Search

Tabu Search (TS) is a heuristic method originally proposed by Glover [5] in 1986. TS has primarily been proposed and developed for combinatorial optimization problems [6, 7, 8], and has shown its capability of dealing with various difficult problems [8, 19]. Moreover, there have been some attempts to develop TS for continuous optimization problems [9].

The main feature of TS is its use of an adaptive memory and responsive exploration. A simple TS combines a local search procedure with anti-cycling memory-based rules to prevent the search from getting trapped in local optimal solutions. Specifically, TS avoids returning to recently visited solutions by constructing a list of them called *Tabu List* (TL). In each iteration of the simple TS illustrated in Algorithm 2.1 below, TS generates many trial solutions in a neighborhood of the current solution. The trial solutions generation process is composed to avoid generating any trial solution that has already been visited recently. The best trial solution in the generated solutions will become the next solution. Therefore, TS can accept downhill movements to avoid getting trapped in local maxima. TS can be terminated if the number of iterations without any improvement exceeds a predetermined maximum iteration number.

Algorithm 2.1 *Simple Tabu Search*

1. *Choose an initial solution x_0. Set the Tabu List (TL) to be empty, and set the counter $k := 0$.*
2. *Generate neighborhood moves list $M(x_k) = \{x' : x' \in N(x_k)\}$, based on tabu restrictions, where $N(x_k)$ is a neighborhood of x_k.*
3. *Let x_{k+1} be the best trial solution in $M(x_k)$, and update TL.*
4. *If stopping conditions are satisfied, terminate. Otherwise, go to Step 2.*

A simple TS structure given in Algorithm 2.1 is called *short-term memory TS*. Updating the memory-based TL can be modified and controlled by the following concepts:

- **Tabu tenure:** the number of iterations in which a move is considered to remain tabu or forbidden;
- **Aspiration criteria:** accepting an improving solution even if generated by a tabu move.

The short-term memory is built to keep the recency only. In order to achieve better performance, long-term memory is used to keep more important search features besides the recency, such as the quality and the frequency. Specifically, long-term memory in high-level TS records solutions of special characters like elite and frequently visited solutions. Then, the search process of TS can adapt itself by using these special types of solutions in the following aspects:

- **Diversification:** generating new solutions by depressing attributes of frequently visited solutions in order to diversify the search to other areas of the solution space.
- **Intensification:** giving priority to elite or promising solutions in order to obtain much better solutions in their vicinity.

3 Tabu Search for Attribute Reduction (TSAR)

In this section, a TS-based method called Tabu Search for Attribute Reduction (TSAR) is proposed to deal with the attribute reduction problem in rough set theory. First we describe the components of TSAR, and then state the TSAR algorithm formally.

3.1 Solution Representation

TSAR uses a binary representation for solutions (attribute subsets). Therefore, a trial solution x is a 0-1 vector with dimension equal to the number of conditional attributes $|\mathbb{C}|$. If a component x_i of x, $i = 1, \ldots, |\mathbb{C}|$, has the value 1, then the i-th attribute is contained in the attribute subset represented by the trial solution x. Otherwise, the solution x does not contain the i-th attribute.

3.2 Solution Quality Measure

The dependency degree $\gamma_x(\mathbb{D})$ of decision attribute $\mathbb{D}$ is used to measure the quality of a solution x, where $\gamma_x(\mathbb{D})$ is identified with $\gamma_P(\mathbb{D})$ such that P contains those attributes corresponding to $x_i = 1$. Comparing two solution x and x', we say x is better than x' if one of the following conditions holds:

- $\gamma_x(\mathbb{D}) > \gamma_{x'}(\mathbb{D})$,
- $\sum_i x_i < \sum_i x'_i$ if $\gamma_x(\mathbb{D}) = \gamma_{x'}(\mathbb{D})$.

3.3 Tabu List

The role of TSAR Tabu List (*TL*) is to avoid being trapped in local solutions and avoid generating useless solutions. Therefore, the first and second positions in *TL* are reserved for these useless solutions which are solution of all ones (i.e., all attributes are considered) and solution of all zeros (i.e., all attributes are discarded). The remaining positions in *TL* , i.e., $|TL| - 2$ positions, are used to save the most recently visited solutions.

3.4 Neighborhood Trials Generation

TSAR invokes a procedure to generate ℓ trial solutions in a neighborhood of a current solution x. Specifically, trial solutions y^j, $j = 1, \ldots, \ell$, are generated by changing j positions in x randomly based on tabu restrictions as in the following procedure.

Procedure 3.1 $[y^1, \ldots, y^\ell] = \text{Trials}(x, \text{TL}, \ell)$

1. *Repeat the following steps for* $j = 1, \ldots, \ell$.
2. *Set* $y^j := x$, *and choose* j *random positions* $p_1, \ldots, p_j$ *of* y^j.
3. *Update the chosen positions by the rule* $y^j_{p_i} := 1 - y^j_{p_i},\ i = 1, \ldots, j$.
4. *If* $y^j \in$ TL, *then return to Step 2 to generate another* y^j.

3.5 Diversification-Intensification Scheme

An intelligent search method should invoke a wide exploration mechanism as well as a deep exploitation mechanism. Even if these mechanisms are well-defined, the most challenge is to apply them in appropriate time to avoid more unneeded complexity or premature convergence. TSAR invokes three mechanisms for Diversification and Intensification. Actually, TSAR checks the search progress to apply these mechanisms gradually in order to improve search progress. The first mechanism is a diversification one which provides the search with a diverse solution if the search cannot find any improvement. TSAR defines a vector v^F of dimension $|\mathbb{C}|$ which counts the numbers of choosing each conditional attribute among the generated trial solutions. Then, a diverse solution x^{div} can be generated to contain attributes chosen with probability inversely proportional to their appearance in v^F. Specifically, Procedure 3.2 states how x^{div} is generated.

Procedure 3.2 $[x^{div}] = \text{Diverse}(v^F)$

1. *Generate random numbers* $r_1, \ldots, r_{|v^F|} \in (0, 1)$.
2. *Repeat the following step for* $i = 1, \ldots, |\mathbb{C}|$.
3. *If* $r_i > v^F_i$, *set* $x^{div}_i := 1$. *Otherwise, set* $x^{div}_i := 0$.

If the search still cannot find any improvement during some iterations after generating x^{div}, TSAR applies an intensification mechanism to refine the best reduct x^{best} found so far. At this stage, TSAR should obtain at least one reduct. Otherwise, the search is restarted again from a new generated x^{div}. Actually, numerical experiments show that TSAR can obtained many reducts without need of repeating the diverse solution procedure. The best reduct x^{best} refinement, called *Shaking*, tries to

reduce the attributes contained in x^{best} one by one without reducing $\gamma_{x^{best}}(\mathbb{D})$ as shown in Procedure 3.3. The search is continued from x^{best} no matter whether it can be improved through the Shaking Mechanism or not. Then, the main TSAR search is terminated and a final refinement is applied.

Procedure 3.3 Shaking(x^{best})

1. *Construct the set W of all positions of ones in x^{best}, i.e., the elements of $W = \{w_1, \ldots, w_{|W|}\}$ represent the attributes contained in x^{best}.*
2. *Repeat the following steps for $j = 1, \ldots, |W|$.*
3. *Delete the attribute $w_j \in W$ by setting $x^{best}_{w_j} := 0$, and compute a γ-value.*
4. *Update x^{best} if γ-value is increased or, if γ-value remains the same but the number of the attributes contained in reducts is decreased.*

The final diversification-intensification mechanism is called *Elite Reducts Inspiration*. In the TSAR process of finding the minimal reduct, any reduct which has been visited is saved in a set called *Reduct Set* (*RedSet*). It is well known that *Core* is the intersection of all the reducts and it is the most important subset of attributes, since none of its elements can be removed without affecting of the classification power of attributes. Here the idea of *Core* is introduced to construct a trial solution x^{ERI} as the intersection of the n_R best reducts in *RedSet* instead of all the reducts, where n_R is a pre-specified number. If the number of attributes involved in x^{ERI} is less than that in x^{best} by at least two, then the zero position in x^{ERI} which gives the highest γ-value is updated to be one. This mechanism is continued until the number of attributes involved in x^{ERI} becomes less than that in x^{best} by one.

Procedure 3.4 $[x^{\text{ERI}}] = \text{EliteReducts}(\text{RedSet}, n_R)$

1. *If* RedSet *is empty, then return. Otherwise, go to Step 2.*
2. *Set n_F equal to the number of attributes involved in the best reduct in* RedSet*, and set x^{ERI} equal to the intersection of the n_R best reducts in* RedSet*.*
3. *If $\sum_{i=1}^{|\mathbb{C}|} x_i^{\text{ERI}} < n_F - 1$, then go to Step 4. Otherwise, return.*
4. *If $\gamma_{x^{\text{ERI}}}(\mathbb{D}) = 1$, then return.*
5. *Update the zero position in x^{ERI} which gives the highest γ-value, and go to Step 3.*

3.6 TSAR Algorithm

TSAR starts with an initial solution x^0 and uses the tabu restriction rule in generating trial solutions in a neighborhood of the current solution. TSAR continues generating trial solutions until no improvement can be obtained through I_{div} consecutive iterations. Then, TSAR proceeds to the search process starting from a diverse solution x^{div} generated by Procedure 3.2. If the number of these consecutive iterations without improvement exceeds I_{shak}, TSAR invokes Procedure 3.3 to improve the best reduct x^{best} obtained so far. Then, the search is continued starting from x^{best}. The search may be terminated if the number of iterations exceeds $I_{\max}$, or the number of consecutive iterations without improvement exceeds I_{imp}. Finally, Elite Reducts Inspiration is applied as a final diversification-intensification mechanism. The main structure of TSAR is shown in Figure 1 and the formal algorithm is stated below.

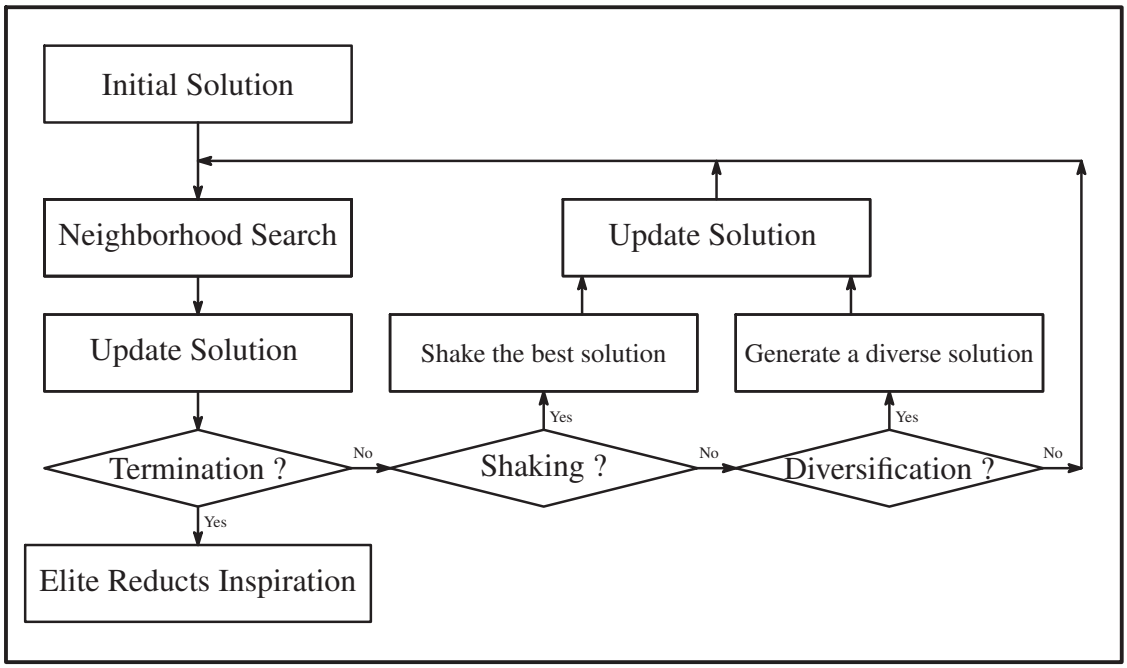

Figure 1: TSAR Flowchart

Algorithm 3.5 TSAR

1. *Let the Tabu List (*TL*) contain the two extreme solutions* $(0, \ldots, 0)$ *and* $(1, \ldots, 1)$*, set* v^F *to be a zero vector, and set* RedSet *to be an empty set. Choose an initial solution* x^0*, and set the counter* $k := 0$*. Select* $I_{\max}$*,* I_{imp}*,* I_{shak} *and* I_{div} *such that* $I_{\max} > I_{imp} > I_{shak} > I_{div}$*.*

2. *Generate neighborhood trials* $y^1, \ldots, y^\ell$ *around* x^k *using Procedure 3.1.*

3. *Set* x^{k+1} *equal to the best trial solution from* $y^1, \ldots, y^\ell$*, and update* TL*,* v^F*,* x^{best} *and* RedSet*. Set* $k := k + 1$*.*

4. *If the number of iterations exceeds* $I_{\max}$*, or the number of iterations without improvement exceeds* I_{imp}*, go to Step 7.*

5. *If the number of iterations without improvement exceeds* I_{shak}*, apply Procedure 3.3 to improve* x^{best}*, set* $x^k := x^{best}$ *and go to Step 2.*

6. *If the number of iterations without improvement exceeds* I_{div}*, apply Procedure 3.2 to obtain a new diverse solution* x^{div}*, set* $x^k := x^{div}$*, and go to Step 2.*

7. *Apply Procedure 3.4 to obtain* x^{ERI}*. Update* x^{best} *by* x^{ERI} *if the latter is better, and terminate.*

4 Numerical Experiments

Algorithm 3.5 was programmed in MATLAB and applied to 13 well-known datasets [11, 12], see Table 2. For each dataset, the TSAR MATLAB code was run 20 times with different initial solutions.

Table 2: Datasets used in the experiments

Dataset	No. of Attributes	No. of Objects
M-of-N	13	1000
Exactly	13	1000
Exactly2	13	1000
Heart	13	294
Vote	16	300
Credit	20	1000
Mushroom	22	8124
LED	24	2000
Letters	25	26
Derm	34	366
Derm2	34	358
WQ	38	521
Lung	56	32

The results are reported in Tables 4 and 5. Before discussing these results, we explain the setting of the TSAR parameters. In Table 3, we summarize all parameters used in TSAR with their assigned values. These chosen values are based on the common setting in the literature or based on our numerical experiments.

TSAR parameters are categorized into three groups:

- *Neighborhood Search Parameters:* ℓ is the size of neighborhood trial solutions, and $|TL|$ is the size of tabu list.
- *Diversification and Intensification Parameters:* I_{div} and I_{shak} are the numbers of non-improvement iterations to apply diversification and shaking, respectively, and n_R is the number of the best reducts used to compute x^{ERI} in Procedure 3.4.
- *Termination Parameters:* I_{max} is the maximum number of iterations, and I_{imp} is the maximum number of non-improvement consecutive iterations.

The performance of TSAR was tested using different values of these parameters. First, the tabu list size $|TL|$ was set to 5 to save the three most recently visited solutions in addition to the all-zeros and all-ones solutions. The preliminary numerical experiments showed that this setting was enough to escape from local solutions. The number ℓ of neighborhood trial solutions was set equal to $round(\frac{|\mathbb{C}|}{2})$. Since l depends on the problem size, it may help avoid a deterioration of the method when the problem size increases. The maximum numbers I_{div} and I_{shak} of consecutive non-improvement iterations before applying diversification or shaking are set equal to 10 and 20, respectively. Actually, the preliminary numerical experiments showed that these settings were reasonable to avoid spending more non-improvement iterations, and sufficient to let the search process explore the current region before supporting it with diversification or intensification. The number n_R of best reducts is set equal to 3, which helps TSAR to improve the best reduct found so far if it was not a global one. Finally, the maximum numbers I_{max} and I_{imp} of iterations and of consecutive non-improvement iterations,

Table 3: TSAR Parameter Setting

Parameter	Definition	Value
$\|TL\|$	size of tabu list	5
ℓ	size of neighborhood	$round(\frac{\|\mathbb{C}\|}{2})$
n_R	number of the best reducts used to compute x^{ERI}	3
I_{max}	max number of iterations	100
I_{imp}	max number of consecutive non-improvement iterations	40
I_{div}	number of consecutive non-improvement iterations to apply diversification	10
I_{shak}	number of consecutive non-improvement iterations to apply shaking	20

respectively, are set as in Table 3. The preliminary numerical experiments showed that these settings were enough to avoid premature termination.

TSAR was compared with three other rough set attribute reduction methods;

- Ant colony optimization for rough set attribute reduction (AntRSAR) [11, 12],
- Simulated annealing for rough set attribute reduction (SimRSAR) [12], and
- Genetic algorithm for rough set attribute reduction (GenRSAR) [11, 12]

These comparisons are reported in Tables 4 and 5. The results of AntRSAR, SimRSAR, and GenRSAR reported in Tables 4 are taken from their original papers [11, 12]. The results in Table 4 represent the numbers of attributes in the minimal reducts obtained by each method, and the superscripts in parentheses represent the number of runs that achieved this number. The number of attributes without superscripts mean that the method could obtain this number of attributes for all runs. All methods have the same number of runs for each dataset, which is 20 except the results of SimRSAR for Heart, Vote and Drem2 datasets for which the number of runs are 30, 30 and 10, respectively, as in [12]. TSAR could obtain the best known minimal reducts for all tested datasets. Moreover, for 6 datasets among them, TSAR could obtain the best known minimal reducts in all runs.

To make a more detailed comparison, the mean values of the numbers of attributes in the best reducts obtained for each dataset are reported in Table 5. Moreover, t-tests [15] for these means are also reported to show the differences among the compared methods at 0.05 level of significance. TSAR outperforms GenRSAR for all tested datasets except for Heart dataset. Moreover, the average results of TSAR for this dataset are still better than those of GenRSAR. SimRSAR could not outperform TSAR for any dataset, while TSAR outperforms it for Vote dataset. The performance of TSAR and AntRSAR is comparable since there is no significant difference between them for 11 datasets out of the 13 tested ones, and TSAR outperforms AntRSAR for LED dataset, while AntRSAR outperforms TSAR for Lung dataset.

It does not seem easy to make a complete comparison of those methods in terms of computational costs and complexities. However, we try to comment on computational costs of calculating the dependency degree function γ, since its computation is costly especially if the dataset has a large number of attributes or objects. First, GenRSAR uses the population size equal to 100, the probabilities of applying the crossover and mutation processes are 0.6 and 0.4, respectively, and the termination is done after 100 generations. Therefore, GenRSAR uses approximately 10,000 γ-function evaluations for each dataset. According to the cooling schedule of SimRSAR, the number of γ-function evaluations

Table 4: Results of TSAR and other methods

Dataset	No. of Attributes	AntRSAR	SimRSAR	GenRSAR	TSAR
M-of-N	13	6	6	$6^{(6)}7^{(12)}$	6
Exactly	13	6	6	$6^{(10)}7^{(10)}$	6
Exactly2	13	10	10	$10^{(9)}11^{(11)}$	10
Heart	13	$6^{(18)}7^{(2)}$	$6^{(29)}7^{(1)}$	$6^{(18)}7^{(2)}$	6
Vote	16	8	$8^{(15)}9^{(15)}$	$8^{(2)}9^{(18)}$	8
Credit	20	$8^{(12)}9^{(4)}10^{(4)}$	$8^{(18)}9^{(1)}11^{(1)}$	$10^{(6)}11^{(14)}$	$8^{(13)}9^{(5)}10^{(2)}$
Mushroom	22	4	4	$5^{(1)}6^{(5)}7^{(14)}$	$4^{(17)}5^{(3)}$
LED	24	$5^{(12)}6^{(4)}7^{(3)}$	5	$6^{(1)}7^{(3)}8^{(16)}$	5
Letters	25	8	8	$8^{(8)}9^{(12)}$	$8^{(17)}9^{(3)}$
Derm	34	$6^{(17)}7^{(3)}$	$6^{(12)}7^{(8)}$	$10^{(6)}11^{(14)}$	$6^{(14)}7^{(6)}$
Derm2	34	$8^{(3)}9^{(17)}$	$8^{(3)}9^{(7)}$	$10^{(4)}11^{(16)}$	$8^{(2)}9^{(14)}10^{(4)}$
WQ	38	$12^{(2)}13^{(7)}14^{(11)}$	$13^{(16)}14^{(4)}$	16	$12^{(1)}13^{(13)}14^{(6)}$
Lung	56	4	$4^{(7)}5^{(12)}6^{(1)}$	$6^{(8)}7^{(12)}$	$4^{(6)}5^{(13)}6^{(1)}$

is about $40M\log(2|\mathbb{C}|/T_{\min})$, where M is the epoch length of the cooling schedule, and $T_{\min}$ is the minimum temperature used to terminate the cooling schedule. AntRSAR spends 250 iterations before termination. In each of the AntRSAR iterations, one ant starts from each attribute, hence, $|\mathbb{C}|$ ants are used in total. Each ant tries to add attributes one by one until a reduct is found, and then the iteration is terminated. For TSAR, the maximum number of γ-function evaluations is equal to $50|\mathbb{C}|$ plus those in Shaking and Elite Reducts Inspiration processes. Actually, TSAR applies Shaking at most once and the maximum cost of γ-function evaluations for applying Shaking is $|\mathbb{C}|-2$. Moreover, TSAR applies Elite Reducts Inspiration once and the numerical experiments showed that this step costs at most $3|\mathbb{C}|$ γ-function evaluations. According to these observations, we can conclude that

- the exact number of γ-function evaluations in GenRSAR is approximately 10,000,
- a lower bound for the number of γ-function evaluations in AntRSAR is $250|\mathbb{C}|(|\mathbb{R}_{\min}|-2)$, where $\mathbb{R}_{\min}$ is the minimal reduct,
- an upper bound for the number of γ-function evaluations in TSAR is $54|\mathbb{C}|$.

Figure 2 shows these estimates of γ-function evaluations in all tested datasets. For GenRSAR, the number of γ-function evaluations in each dataset is around the number represented by "$\square$". The number of γ-function evaluations in AntRSAR is not less than those represented by "$\triangle$". Finally, the number of γ-function evaluations in TSAR does not exceed those represented by "$\triangledown$". It is clear that TSAR is much cheaper than GenRSAR and AntRSAR. Moreover, from Figure 5 in [12], the average runtime for SimRSAR is worse than GenRSAR for all datasets except LED. Therefore, we may conclude that TSAR is generally cheaper than SimRSAR, too.

Table 5: Comparison between TSAR and other methods

Dataset	Solution Qualities (Averages)				t-value (Significant Method at level 0.05)		
	AntRSAR	SimRSAR	GenRSAR	TSAR	AntRSAR – TSAR	SimRSAR – TSAR	GenRSAR – TSAR
M-of-N	6	6	6.6667	6	–	–	5.83 (TSAR)
Exactly	6	6	6.5	6	–	–	4.36 (TSAR)
Exactly2	10	10	10.55	10	–	–	4.82 (TSAR)
Heart	6.1	6.0333	6.1	6	1.45	1.00	1.45
Vote	8	8.5	8.9	8	–	5.38 (TSAR)	13.08 (TSAR)
Credit	8.6	8.2	10.7	8.45	0.63	–1.14	12.09 (TSAR)
Mushroom	4	4	6.65	4.15	–1.83	–1.83	16.15 (TSAR)
LED	5.5263	5	7.75	5	2.97 (TSAR)	–	22.36 (TSAR)
Letters	8	8	8.6	8.15	–1.83	–1.83	3.24 (TSAR)
Derm	6.15	6.4	10.7	6.3	–1.12	0.65	29.59 (TSAR)
Derm2	8.85	8.7	10.8	9.1	–1.68	–2.04	11.05 (TSAR)
WQ	13.45	13.2	16	13.25	1.02	–0.33	22.36 (TSAR)
Lung	4	4.7	6.6	4.75	–6.10 (AntRSAR)	–0.28	11.10 (TSAR)

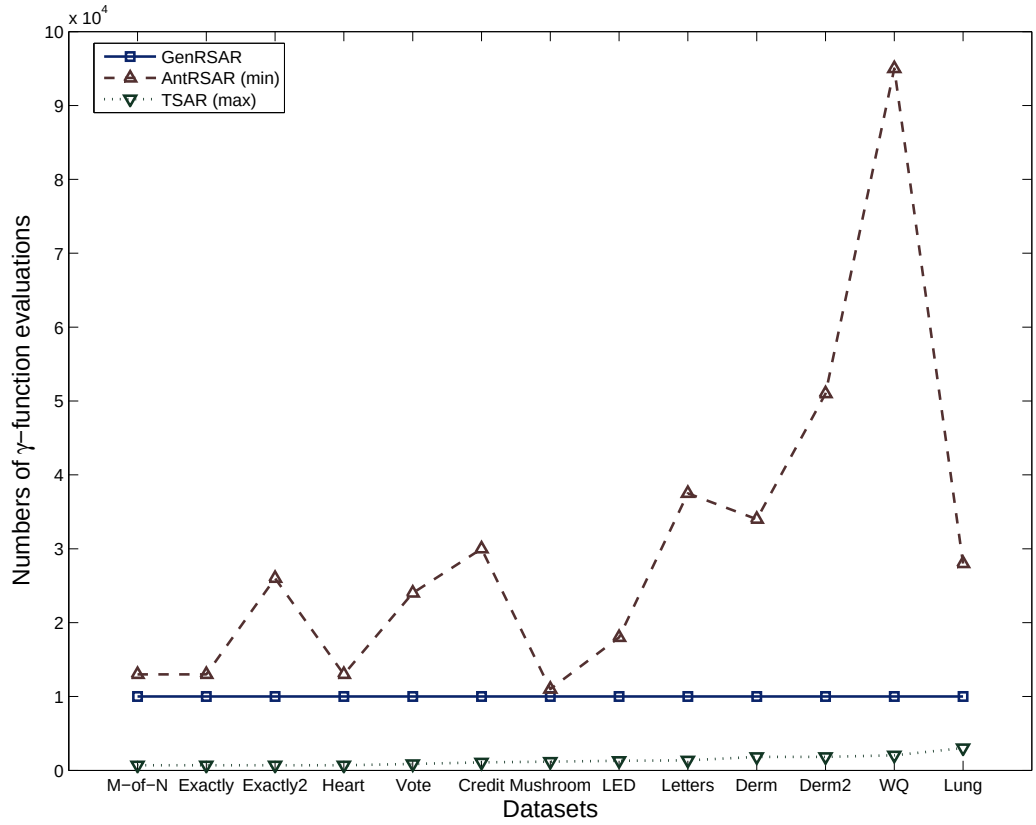

Figure 2: Costs of computing the dependency degree function γ

5 Conclusion

The attribute reduction problem in rough set theory has been studied in this paper. A TS-based method, called Tabu Search for Attribute Reduction (TSAR), has been proposed to solve the problem. New diversification and intensification elements have been inlaid in TSAR to achieve better performance and to fit the problem. Numerical experiments on 13 well-known datasets have been presented to show the efficiency of TSAR. Comparisons with other CI tools have revealed that TSAR is promising and it is less expensive in computing the dependency degree function γ.

Acknowledgment

This work was supported in part by the Scientific Research Grant-in-Aid from Japan Society for the Promotion of Science, and also by the Information Research Center for Development of Knowledge Society Infrastructure, Graduate School of Informatics, Kyoto University.

References

[1] A. Bargiela, W. Pedrycz (2002) Granular Computing: An Introduction. Springer-Verlag, Berlin.

[2] E.K. Burke, G. Kendall (2005) Search Methodlogies: Introductory Tutorials in Optimization and Decision Support Techniques. Springer-Verlag, Berlin.

[3] A. Chouchoulas, Q. Shen (2001) Rough set-aided keyword reduction for text categorisation. Applied Artificial Intelligence, vol. 15, pp. 843–873.

[4] A.P. Engelbrecht (2003) Computational Intelligence: An Introduction. John Wiley & Sons, Chichester, England.

[5] F. Glover (1986) Future paths for integer programming and links to artificial intelligence. Computers and Operations Research, 13:533–549.

[6] F. Glover (1989) Tabu search–Part I. ORSA Journal on Computing, vol. 1, pp. 190–206.

[7] F. Glover (1990) Tabu search–Part II. ORSA Journal on Computing, vol. 2, pp. 4–32.

[8] F. Glover, M. Laguna (1997) Tabu Search. Kluwer Academic Publishers, Boston, MA, USA.

[9] A. Hedar, M. Fukushima (2006) Tabu search directed by direct search methods for nonlinear global optimization. European Journal of Operational Research, vol. 170, pp. 329–349.

[10] J. Jelonek, K. Krawiec, R. Slowinski (1995) Rough set reduction of attributes and their domains for neural networks. Computational Intelligence vol. 11, pp. 339–347.

[11] R. Jensen, Q. Shen (2003) Finding rough set reducts with ant colony optimization. Proceedings of the 2003 UK Workshop on Computational Intelligence, pp. 15–22.

[12] R. Jensen, Q. Shen (2004) Semantics-preserving dimensionality reduction: Rough and fuzzy-rough-based approaches. IEEE Transactions on Knowledge and Data Engineering, vol. 16, pp. 1457–1471.

[13] A. Konar (2005) Computational Intelligence: Principles, Techniques and Applications. Springer-Verlag, Berlin.

[14] T.Y. Lin, Y.Y. Yao, L.A. Zadeh (2002) Data Mining, Rough Sets and Granular Computing. Springer-Verlag, Berlin.

[15] D.C. Montgomery, G.C. Runger (2003) Applied Statistics and Probability for Engineers. Thrid Edition. John Wiley & Sons, Chichester, England.

[16] Z. Pawlak (1982) Rough sets. International Journal of Computer and Information Sciences, vol. 11, pp. 341–356.

[17] Z. Pawlak (1991) Rough Sets: Theoretical Aspects of Reasoning about Data. Kluwer Academic Publishers, Dordrecht, The Netherlands.

[18] Z. Pawlak, A. Skowron (2000) Rough set methods and applications: New developments in knowledge discovery in information systems. Studies in Fuzziness and Soft Computing, L. Polkowski, T.Y. Lin, and S. Tsumoto (eds.), vol. 56, Physica-Verlag, Berlin.

[19] C. Rego, B. Alidaee (2005) Metaheursitic Optimization via Memory and Evolution. Springer-Verlag, Berlin.

[20] R.W. Swiniarski, A. Skowron (2003) Rough set methods in feature selection and recognition. Pattern Recognition Letters, vol. 24, pp. 833–849.

[21] S. Tan (2004) A global search algorithm for attributes reduction. *AI 2004: Advances in Artificial Intelligence*, G.I. Webb and X. Yu (eds.), LNAI 3339, pp. 1004–1010.

[22] A. Tettamanzi, M. Tomassini, J. Janben (2001) Soft Computing: Integrating Evolutionary, Neural, and Fuzzy Systems. Springer-Verlag, Berlin.

[23] L.-Y. Zhai, L.-P. Khoo, S.-C. Fok (2002) Feature extraction using rough set theory and genetic algorithms– an application for simplification of product quality evaluation. Computers & Industrial Engineering, vol. 43, pp. 661–676.