

Parallel Feature Selection Algorithm based on Rough Sets and Particle Swarm Optimization

Mateusz Adamczyk

Faculty of Mathematics, Informatics, and Mechanics

University of Warsaw

Banacha 2

02-097 Warszawa

Poland

Email: adamczyk@mimuw.edu.pl

Abstract—The aim of this paper is to propose a new method of solving feature selection problem. Foundations of presented algorithm lie in the theory of rough sets. Feature selection methods based on rough sets have been used with success in many data mining problems, but their weakness is their computational complexity. In order to overcome the above-mentioned problem, researchers used diverse approximation techniques. This paper presents a new approach to approximation of reducts.

Particle swarm optimization (PSO) is a stochastic meta-heuristic similar to genetic algorithms. The idea is to see each potential solution as a particle with certain velocity flying through the problem space. The PSO finds optimal solutions by interactions of individuals in population. The main advantage of the PSO over genetic algorithms, is that PSO does not require complex operators such as crossover or mutation. It only uses simple mathematical operators to update position and velocity of each particle, which makes PSO computationally inexpensive in terms of both memory and runtime.

The presented feature selection algorithm treats each feature subset as separate particle. Optimal subset, in terms of selected measure, is discovered as particles fly within the problem space. In order to speed up calculations and balance usage of hardware resources (processors, memory), parallel asynchronous version of PSO is applied. It is based on scheduling calculations of complex fitness function on slave processors, while the main one is responsible for updating particles data and checking algorithm's convergence. Applied approach scales well and provides balanced usage of given resources even if it is not feasible to use the same computational power of every processor, for instance when used resources are not homogeneous.

Proposed method was tested on selected set of data sets from the UCI repository and results were compared to some of the classical algorithms.

Index Terms—Feature Selection, Rough Sets, Particle Swarm Optimization, Parallel Asynchronous Particle Swarm Optimization

I. INTRODUCTION

FEATURE selection is one of phases in data mining. The idea is to select subset of attributes, which preserves knowledge for given information or decision system. There are two main reasons for doing it. Firstly, it is some kind of data

This research was partly supported by Polish National Science Centre (NCN) grants DEC-2011/01/B/ST6/03867 and DEC-2012/05/B/ST6/03215.

compression which eases comprehension of analysed data. Secondly, most classifiers are better trained on non-redundant data (compare with [6]). Moreover, removal of superfluous attributes leads to smaller data, which will quicken classifier training.

There are many algorithms for feature selection. Some of them are simple filters concerning every attribute separately. The more useful ones try to rank sets of attributes. They have also high computational complexity (see [1]). One of methods for finding „good” subset of attributes is calculating reducts. It is based on rough sets theory. The reduct is a set of attributes which: preserve information contained in full set of attributes and is minimal, i.e. removing one attribute from reduct leads to losing some knowledge about data. Unfortunately, calculating reducts is quite complex [7].

Feature selection algorithms can be divided into two groups: *filters* and *wrappers* ([2] and [3]). The most standard filters select attributes based on some measure and are classifier agnostic. On the other hand, wrappers use selected classifier accuracy as a measure of quality of subset of attributes. By doing that, they are somewhat „tied” to used classifier and their results will probably not work well with other learning algorithms.

The aim of this paper is to present a new feature selection algorithm. It is wrapper algorithm, that uses classical, exhaustive method for finding reducts and inducing decision rules from them. Because reducts are induced on a subset of attributes, time of their calculations should be smaller than time for obtaining reducts from all attributes. Finding new candidates for reducts is done by using particle swarm optimization. The idea was inspired by work presented in [4]. Particle swarm optimization is meta-heuristic originally developed as a simulation of birds flocking. Because it is a little simpler to apply and implement than genetic algorithms, it has gained researchers attention lately. In order to further speed up the approximation, parallel asynchronous version of particle swarm optimization ([14]) was applied.

The rest of this paper is divided as follows. Section II contains basic information about rough sets theory and reducts calculations. In section III particle swarm optimization has been presented. Above-mentioned section also contains com-

parison of synchronous and asynchronous versions of this meta-heuristic. Subsection III-C of section III contains detailed information about algorithm presented in [4]. Proposed algorithm for feature selection is described in section IV, where it is also compared with method from [4]. In the same section, experimental results: comparison of reducts and found attributes subsets, as well as parallelization gains are also presented. Section V contains conclusions and possible extensions to presented research.

II. ROUGH SETS

The development of rough sets theory was started by professor Z. Pawlak in 1981 [5]. Its main purpose is to deal with uncertainty of information or decision systems. *The information system* \mathcal{A} is a pair of a non-empty, finite set of objects U called *universe* and a non-empty, finite set of their *attributes* A (see equation 1).

$$\mathcal{A} = (U, A) \quad (1)$$

An attribute is a function $a : U \rightarrow V_a, \forall a \in A$, where set V_a is called a value set of a . *The decision system* is an information system extended by one distinguished attribute $d \notin A$ called *decision*.

As objects in information system are described only by attributes, two cases can occur:

- different objects can have the same values on all attributes, or
- some attributes can be superfluous.

To deal with the former case, an indiscernibility relation is used. More formally:

$$IND_{\mathcal{A}}(B) = \{(x, x') \in U^2 : \forall a \in B a(x) = a(x')\} \quad (2)$$

The $IND_{\mathcal{A}}(B)$ from equation 2 is called *B-indiscernibility relation*. If $(x, x') \in IND_{\mathcal{A}}(B)$, then objects x and x' are indiscernible from each other by attributes from B . The equivalence classes of the B-indiscernibility relation are denoted $[x]_B$.

For a subset of objects $X \in U$ and a subset of attributes $B \in A$, X can be approximated only by attributes from B by constructing its *lower approximation* $\underline{B}X = \{x : [x]_B \subseteq X\}$ and its *upper approximation* $\overline{B}X = \{x : [x]_B \cap X \neq \emptyset\}$. The objects in $\underline{B}X$ are certainly in X basing on knowledge in B , whereas objects from $\overline{B}X$ can be classified only as possible elements of X on the basis of knowledge in B . The set $BN_B(X) = \overline{B}X \setminus \underline{B}X$ is called a *B boundary region of X*, and consists of objects which we cannot decisively classify into X on the basis of knowledge in B . A set is said to be *rough* (respectively *crisp*) if the boundary region is non-empty (respectively empty).

In order to speed up classification and clarify knowledge about data, redundant attributes can be removed. To do so, one can keep only those attributes, which preserve indiscernibility relation and hence set approximation. Rejected attributes were redundant since their removal has not worsened classification accuracy. There are usually several such subsets and those

which are minimal in terms of cardinality are called *reducts*. It can be shown that the number of reducts of an information system with m attributes may be equal to $\binom{m}{\lfloor \frac{m}{2} \rfloor}$. Moreover, finding minimal reduct, i.e. a reduct the cardinality of which is the smallest among other reducts, is NP-hard [7].

In my experiments I have used classical algorithm for calculating reducts. Its detailed description can be found in [7]. The algorithm is executed in two steps: calculating a *discernibility matrix* and finding all prime implicants of a *discernibility function* induced from the discernibility matrix. Obtaining decision rules from reducts is straightforward: for each pair of object and reduct those attributes and their values are taken from object, that are also in reduct. Such pairs create conditional part of the rule. The decision part is made from decision for analysed object.

For the information system \mathcal{A} with n objects, the *discernibility matrix* is a symmetric $n \times n$ matrix with entries given in equation 3.

$$\forall_{i,j \in \{1,2,\dots,n\}} c_{ij} = \{a \in A : a(x_i) \neq a(x_j)\} \quad (3)$$

The *discernibility function* $f_{\mathcal{A}}$ for an information system \mathcal{A} is a Boolean function of m Boolean variables $a_1^*, a_2^*, \dots, a_m^*$, which correspond to attributes a_1, a_2, \dots, a_m , defined as on equation 4, where $c_{ij}^* = \{a^* : a \in c_{ij}\}$.

$$f_{\mathcal{A}}(a_1^*, a_2^*, \dots, a_m^*) = \bigwedge \left\{ \bigvee c_{ij}^* : i, j \in \{1, 2, \dots, n\} \wedge c_{ij}^* \neq \emptyset \right\} \quad (4)$$

The set of all prime implicants determines the set of all reducts of \mathcal{A} .

III. PARTICLE SWARM OPTIMIZATION

Particle swarm optimization is a stochastic meta-heuristic developed by Eberhart and Kennedy in 1995 [11]. It was originally created to graphically model behaviour of bird flocking or fish schooling. Initial simulations were transformed into optimization algorithm, and later enhanced by introducing inertia weight [12].

A. Synchronous Particle Swarm Optimization

Particle swarm optimization is an algorithm similar to genetic algorithm [15]. In both cases solutions are mapped into parts of population: in case of particle swarm optimization population consists of particles, whereas in genetic algorithms there are individuals or phenotypes who form population. In both cases near-optimal solution is found as an individual which is the best fitted one, where fitness measure is the optimized function. Individuals in a new population are created by interactions between parts of the previous one.

Particle swarm optimization is initialized with a random set $P = \{p_1, p_2, \dots, p_k\}$ of particles. Each particle $p_i, i \in \{1, 2, \dots, k\}$ has:

- a position x_i in S dimensional space,
- a velocity v_i ,
- memory of personal best position $best_i$.

There is also stored position $best_g$ of the best particles found so far.

In each population, all particles' positions are updated with formula 5 and particles' new fitnesses are calculated.

$$x_i(t+1) = x_i(t) + v_i(t) \quad (5)$$

Best positions: $best_g$ and $best_i$ are updated if necessary. Particles accelerate according to a formula 6.

$$\begin{aligned} v_i(t+1) = & w(t) \cdot v_i(t) + \\ & c_1 \cdot rand_1() \cdot (best_i - x_i) + \\ & c_2 \cdot rand_2() \cdot (best_g - x_i) \end{aligned} \quad (6)$$

Velocities cannot be larger than some constant v_{max} . If they were, particles would fly too fast and probably miss subspaces containing optimal solutions. The v_{max} constant should be large enough, to allow particles to escape regions with sub-optimal solutions.

The $rand_1$ and $rand_2$ are uniformly distributed random functions in $[0, 1]$. Algorithm's parameters: c_1 and c_2 define particles' acceleration constants. Their high values correspond to high attraction of past sub-optimal solutions, whereas low values allow particles to roam far from target regions. The c_1 constant corresponds to personal best solution, and the c_2 determines how firmly particle follows the flock.

The w in equation 6 is called inertia [12]. It is positive linear function of time. Choosing proper inertia is crucial for providing balance between local and global exploration, thus to ensuring that optimal solution is found in small number of iterations.

In equation 6 one can distinguish three parts. The first one corresponds to particle's „memory”. The second one, controlled with c_1 constant, is linked to particle's „cognition”. The third part, which is governed by constant c_2 , describes „social” behaviour of particle. It is responsible for collaboration among particles.

When all particles in the population are updated, particle swarm optimization algorithm checks its convergence. If the best solution found so far is good enough, then calculations are stopped. If found solution is not good enough, then the whole process of moving particles and obtaining their statistics is repeated.

B. Asynchronous Particle Swarm Optimization

In order to speed up particle swarm optimization algorithms, there were proposed many parallelization strategies. Some of them were based on communication strategies similar to ones used with genetic algorithms [13]. In [14] authors proposed significant change to particle swarm optimization algorithm: asynchrony. By making algorithm asynchronous, authors made it converge faster to some optimal solution. It is worth noting, that above-mentioned change created algorithm, whose results will probably be different from the ones obtained from the classical version.

In the asynchronous version of particle swarm optimization, after obtaining fitness for one particle, convergence check and updates are done. It leads to dynamically updated global best

position, which can be modified after updating one particle and not after updating whole population as in synchronous version of algorithm.

It is noteworthy that asynchronous version of particle swarm optimization running sequentially will produce different results than classical particle swarm optimization. The cause of that difference is the above-mentioned difference in strategies for updating data of global best particle. If the global best position would be updated in the middle of processing one population, then the rest of population would move differently in asynchronous than in the synchronous version of particle swarm optimization.

Parallelization of asynchronous particle swarm optimization is straightforward, when done in master-slave architecture. The master processor is responsible for updating particles, checking convergence and scheduling fitness calculations on slave processors. Slave processor evaluates fitness of given particle's position and returns obtained value to master processor. Communication between master processor and slave ones is done with use of first-in-first-out task queue. As master processor process one particle, the one from the front of the task queue, at time and later schedules its data to slave processor, dynamic load balancing is done implicitly. If some slave processor is slower or more loaded, then it will calculate fitness slower than other ones. It will lead to scheduling more work on faster or less busy processors, because they will more often get tasks from master processor.

C. Feature Selection using Particle Swarm Optimization and Rough Sets

In [4], feature selection algorithm based on particle swarm optimization and rough sets theory was presented. In order to exploit particle swarm optimization for finding relevant attributes, some adaptations were necessary.

1) *Representation of Position:* For decision system with m attributes particle position was coded as a binary bit string of length m . If i -th attribute ($i \in \{1, 2, \dots, m\}$) was chosen, then i -th bit of position string was set to 1. Otherwise, it was set to 0. More formally, there was defined bidirectional mapping from a power set $\mathcal{P}(A)$ of the set of attributes A into space of binary string of length m : $\mathcal{M} : \mathcal{P}(A) \rightarrow \{0, 1\}^m$, such that for $R \subseteq A$, the condition from equation 7 holds.

$$\forall_{i \in \{1, 2, \dots, m\}} \mathcal{M}(R)_i = \begin{cases} 0 & a_i \notin R \\ 1 & a_i \in R \end{cases} \quad (7)$$

The movement of particle corresponds to modifying subset of attributes in order to find a better subset. If i -th bit of particle's position was set from zero to one, then the i -th attribute was added to subset. If i -th bit was set to zero, then the i -th attribute was removed. If proper representation of velocity and fitness measure were chosen, then particles flying towards the best position will correspond to finding subsets of A with most relevant attributes.

2) *Representation of Velocity:* The speed of particle was represented as a positive integer, varying from 1 to v_{max} . Value of the velocity shows how many bits of particle's position

should be changed in the particular moment of time to be the same as in the global best position. In other words, particles fly through problem space towards the current best position.

In order to update particles' speeds and positions, the authors of [4] proposed notion of positions difference. The *difference of positions* is equal to component-wise difference of positions seen as two vectors in m -dimensional space. For instance, if $a = [1, 0, 1, 0, 1]$ and $b = [1, 1, 0, 0, 0]$, then: $a - b = [0, -1, 1, 0, 1]$. The 1's in difference correspond to those bits, which should be set in b to make b equal to a . Similarly, the -1 's denote which bits in b should be unset. To use positions' difference in equation 6, it should be converted to integer. To do so, the authors of [4] proposed sum of all difference's components. In the previous example, it is: $|a - b| = \sum_{j=1}^m (a - b)_j = 1$. If updated speed was smaller than one, it was set to 1.

Formulae for updates of particle's velocity is presented on equation 8. The $i \in \{1, 2, \dots, k\}$ on equation 8 denotes particle's index.

$$v_i(t+1) = \min(v_{max}, \max(1, w(t) \cdot v_i(t) + c_1 \cdot rand_1() \cdot \sum_{j=1}^m (best_i - x_i)_j + c_2 \cdot rand_2() \cdot \sum_{j=1}^m (best_g - x_i)_j)) \quad (8)$$

As it is mentioned above, particle's position is updated to move particle towards the global best position. The two cases are possible:

- 1) $v_i \leq |best_g - x_i|$, $i \in \{1, 2, \dots, k\}$,
- 2) $v_i > |best_g - x_i|$, $i \in \{1, 2, \dots, k\}$.

In the first case, v_i random bits, which are different than the ones in $best_g$, are changed. That way, particle flies towards the global best position, but doing random search instead simply being the same as the best. In the second case, apart from flipping all of the bits which are different from the ones in the $best_g$, the $v_i - |best_g - x_i|$ the similar ones are also flipped. It can be interpreted as a particle flying past the best position and exploring more regions.

3) *Fitness function*: The fitness function used in [4] is presented on equation 9.

$$\mathcal{F}(p_i) = \alpha \cdot \gamma_R(d) + \beta \cdot \frac{m - |R|}{m} \quad (9)$$

The α and β are two parameters corresponding to the importance of classification and subset length, $\alpha \in [0, 1]$, $\beta = 1 - \alpha$. The $\gamma_R(d)$ is classification quality of condition attribute set R , relative to decision d , $|R|$ is its cardinality and $m = |A|$.

To measure classification quality of a condition attribute set R , the authors of [4] used the LEM2 algorithm ([8]) for inducing rules from set R and rule negotiation in classification ([9]). The final score was obtained by doing ten-fold cross validation.

IV. PROPOSED ALGORITHM AND CONDUCTED EXPERIMENTS

A. Proposed algorithm

Proposed algorithm is a fusion of two above-mentioned methods. It is slightly changed algorithm presented in [4], which is shortly described in section III-C. Instead of the LEM2 algorithm, exhaustive algorithm for finding reducts (see section II) has been used. Encoding particle's position, velocity and their update strategies was the same, as presented in section III-C1 and III-C2. The fitness function was also the same as the one shown in section III-C3.

The main advantage of proposed method over the one presented in [4], is usage of asynchronous particle swarm optimization. Every time particle fitness is obtained, that particle data are updated. After update, algorithm checks convergence and, if necessary, updates the global best position. Because of frequent updates, particles react more dynamically to finding new best solution. But the main trait of proposed change is to allow exploitation of parallel architecture of modern processors. I have used parallel asynchronous particle swarm optimization [14] and fused it with algorithm presented in [4].

B. Experimental results

1) *Experimental setting*: The sixth version of the Java language was chosen as an implementation language for proposed algorithm. The Rseslib library was used as a source of implementation of exhaustive algorithm for reducts calculation, inducing classifier (see section II for algorithm's details) and doing cross validation. As in [4], ten-fold cross validation was used. The parallel asynchronous particle swarm optimization [14] was implemented within Data Mining EXpressions Library (dmexl), which provides framework for implementing parallel data mining algorithms, especially the ones for feature selection. The dmexl library is being developed by the author of this paper.

Most of the algorithm's parameters were set to be equal to ones presented in [4]. The α was set to 0.9, and β – to 0.1, as in [4]. See section III-C3 and equation 9 for detailed description of above-mentioned parameters. Acceleration constants: c_1 and c_2 were both set to 2. As in [4], the v_{max} parameter was chosen to be equal to $\frac{m}{3}$. The value of minimal fitness, which could stop algorithm execution before reaching requested number of populations, was set to 0.85.

The inertia weight (see equation 6) was the same as in [4] and it is presented on equation 10.

$$w(t+1) = (w(t) - 0.4) \cdot \frac{P_{no} - t}{P_{no} + 0.4} \quad (10)$$

The P_{no} in equation 10 is the number of populations to simulate. The inertia weight is linear function which decreases with time, and varies between 1.4 and 0.4.

<http://www.oracle.com/technetwork/java/javase/overview/index-jsp-136246.html>

<http://rseslib.mimuw.edu.pl/>

<https://github.com/mateka/dmexl>

Experiments were conducted on a personal computer equipped with: Intel Core i7-4700HQ quad core CPU and 32GB of RAM. Amount of runtime memory available to java virtual machine was limited to 6GB.

2) *Used data tables*: Tests were conducted on set of fifteen data tables from UCI repository [16]. Selected tables are listed on table I. Statistics for selected tables are presented on table II.

TABLE I
DATA TABLES USED IN EXPERIMENTS

Table name	URI
Balloon 1	http://archive.ics.uci.edu/ml/datasets/Balloons
Balloon 2	http://archive.ics.uci.edu/ml/datasets/Balloons
Balloon 3	http://archive.ics.uci.edu/ml/datasets/Balloons
Balloon 4	http://archive.ics.uci.edu/ml/datasets/Balloons
Hayes-Roth	http://archive.ics.uci.edu/ml/datasets/Hayes-Roth
Voting	https://archive.ics.uci.edu/ml/datasets/ Congressional+Voting+Records
Lenses	http://archive.ics.uci.edu/ml/datasets/Lenses
Lung Cancer	http://archive.ics.uci.edu/ml/datasets/Lung+Cancer
Monk 1	http://archive.ics.uci.edu/ml/datasets/MONK's+ Problems
Monk 2	http://archive.ics.uci.edu/ml/datasets/MONK's+ Problems
Monk 3	http://archive.ics.uci.edu/ml/datasets/MONK's+ Problems
Postoperative	https://archive.ics.uci.edu/ml/datasets/ Post-Operative+Patient
Promoters	http://archive.ics.uci.edu/ml/datasets/Molecular+ Biology+%28Promoter+Gene+Sequences%29
Tic Tac Toe	https://archive.ics.uci.edu/ml/datasets/Tic-Tac-Toe+ Endgame
Zoo	http://archive.ics.uci.edu/ml/datasets/Zoo

TABLE II
BASE STATISTICS FOR DATA TABLES USED IN EXPERIMENTS

Table name	Attributes	Objects	Decision classes
Balloon 1	4	16	2
Balloon 2	4	20	2
Balloon 3	4	20	2
Balloon 4	4	20	2
Hayes-Roth	5	132	3
Voting	16	435	2
Lenses	4	24	3
Lung Cancer	56	32	3
Monk 1	7	432	2
Monk 2	7	432	2
Monk 3	7	432	2
Postoperative	8	90	3
Promoters	58	106	2
Tic Tac Toe	9	958	2
Zoo	17	101	7

For all data tables three experiments were made in RSES (see [10]). All of them were ten-fold cross validation classifications. In the first case, exhaustive reducts calculation algorithm was used. There was no rule shortening and conflicts were resolved by „simple voting”. This case is denoted by *Exhaustive 1* on Table III. Two other experiments used rule shortening with rule shortening ratio set to 1.0 and conflicts were resolved by „standard voting”. These are a default

settings in RSES. Algorithms used with these settings were: exhaustive algorithm (*Exhaustive 2*) and LEM2 algorithm. All obtained classification accuracies are presented on table III. Even with 16GB of RAM, RSES was unable to calculate

TABLE III
RSES ALGORITHM ACCURACIES ON USED DATA TABLES

Table name	Exhaustive 1	Exhaustive 2	LEM2
Balloon 1	0.40	0.70	0.40
Balloon 2	0.70	1.00	1.00
Balloon 3	0.60	1.00	1.00
Balloon 4	0.60	1.00	1.00
Hayes-Roth	0.79	0.88	0.93
Voting	0.88	0.95	0.98
Lenses	0.45	0.80	0.85
Lung Cancer	-	-	0.40
Monk 1	0.53	1.00	0.99
Monk 2	0.43	0.49	0.71
Monk 3	0.97	1.00	1.00
Postoperative	0.49	0.42	0.42
Promoters	-	-	0.92
Tic Tac Toe	0.63	0.98	1.00
Zoo	0.73	0.97	1.00

reducts and rules for Lung Cancer and Promoters tables when using exhaustive algorithms, with or without shortening obtained rules.

3) *Algorithm's accuracy*: Proposed algorithm was executed sixty times on each data table. Half of experiments were sequential and thirty were parallel. In each algorithm's execution, there were 20 particles and at most 100 populations. Statistics for obtained accuracies are presented on table IV. The first

TABLE IV
STATISTICS FOR ACCURACIES OF SEQUENTIAL AND PARALLEL VERSIONS OF PROPOSED ALGORITHM

Table name	Sequential			Parallel		
	min	max	avg	min	max	avg
Balloon 1	0.00	0.75	0.73	0.00	0.75	0.74
Balloon 2	0.00	0.80	0.80	0.00	0.80	0.79
Balloon 3	0.00	0.80	0.80	0.00	0.80	0.80
Balloon 4	0.00	0.80	0.80	0.00	0.80	0.80
Hayes-Roth	0.00	0.58	0.56	0.00	0.58	0.56
Voting	0.59	0.96	0.94	0.60	0.96	0.95
Lenses	0.00	0.77	0.71	0.00	0.77	0.71
Lung Cancer	0.05	0.58	0.47	0.03	0.58	0.45
Monk 1	0.00	0.75	0.64	0.00	0.75	0.68
Monk 2	0.00	0.67	0.67	0.00	0.67	0.67
Monk 3	0.00	0.81	0.81	0.00	0.81	0.81
Postoperative	0.00	0.71	0.71	0.00	0.71	0.71
Promoters	0.22	0.71	0.66	0.21	0.78	0.65
Tic Tac Toe	0.00	0.70	0.69	0.00	0.70	0.69
Zoo	0.00	0.61	0.61	0.00	0.61	0.61

thing to note, is that both: sequential and parallel version of algorithm have roughly the same accuracies (see Figure 1). Unfortunately, their accuracies are much lower than those of LEM2 from RSES. On average, proposed algorithm had more stable accuracy than exhaustive one from RSES. It is probably due to working on subsets of attributes. It is also important to remember, that in RSES rules were shortened whereas in proposed algorithm they were not. It probably led to overfitting rules in experiments with proposed algorithm.

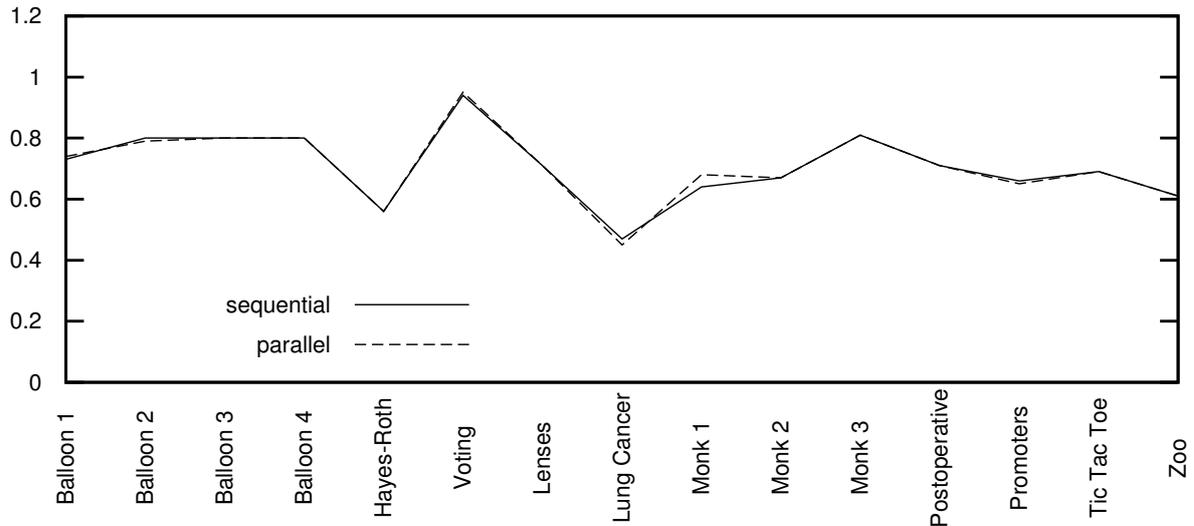


Fig. 1. Accuracies of sequential and parallel versions of proposed algorithm

The interesting conclusion can be drawn from comparing results of exhaustive algorithm without rule shortening and the proposed one. The proposed algorithm performed in all except three cases better than the exhaustive algorithm without rule shortening from RSES. Probably, due to working on smaller sets of attributes, which lead to creation of shorter reducts and rules. The next step in my research is testing proposed algorithm with rule shortening.

Comparison of obtained accuracies from proposed algorithm, exhaustive one from RSES with turned off rules shortening and LEM2 with default settings is presented on Figure 2.

It seems that number of decision classes had no impact on proposed algorithm accuracy, as it was quite stable between different experiments.

TABLE V
SPEEDUP AND EFFICIENCY

Table name	Speedup	Efficiency
Balloon 1	0.59	0.15
Balloon 2	0.58	0.15
Balloon 3	0.58	0.15
Balloon 4	0.58	0.15
Hayes-Roth	1.11	0.28
Voting	19.00	4.75
Lenses	0.60	0.15
Lung Cancer	1.45	0.36
Monk 1	3.76	0.94
Monk 2	3.97	0.99
Monk 3	3.93	0.98
Postoperative	1.00	0.25
Promoters	3.23	0.81
Tic Tac Toe	4.58	1.15
Zoo	1.74	0.43

4) *Algorithm's performance*: For execution times: T_1 – sequential and T_c – parallel on c processors, *speedup* $S_c = \frac{T_1}{T_c}$ and *efficiency* $E_c = \frac{T_1}{T_c \cdot c}$ are measures for expressing parallelization gains. Those measures for proposed algorithm are

shown in table V. The presented values are mean values of all executions for each table. It is worth noting, the number of threads of execution for parallel version of algorithm was set to $4 \cdot c$ in order to fully utilize a given processor. That is the reason why efficiency may be greater than 1. There was an interesting case with Voting table. The sequential version of proposed algorithm have reached maximum number of populations four times, whereas the parallel version reached it only one. That is the reason why speedup for Voting table is so enormous.

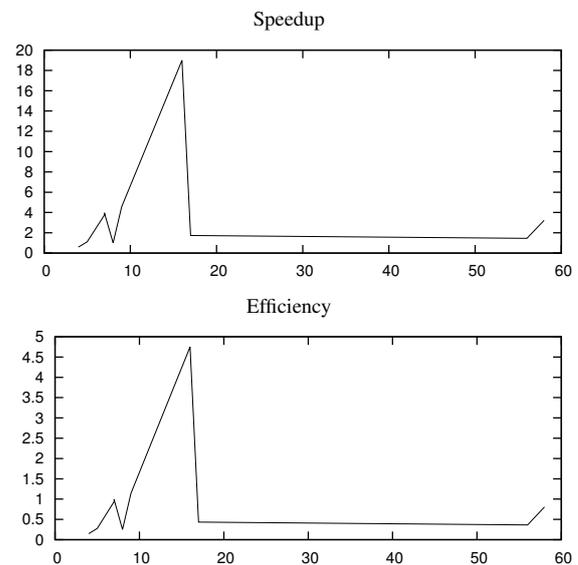


Fig. 3. Relationship between number of attributes and speedup or efficiency

The relationship between the number of attributes and speedup or efficiency is shown on Figure 3. X-axis shows

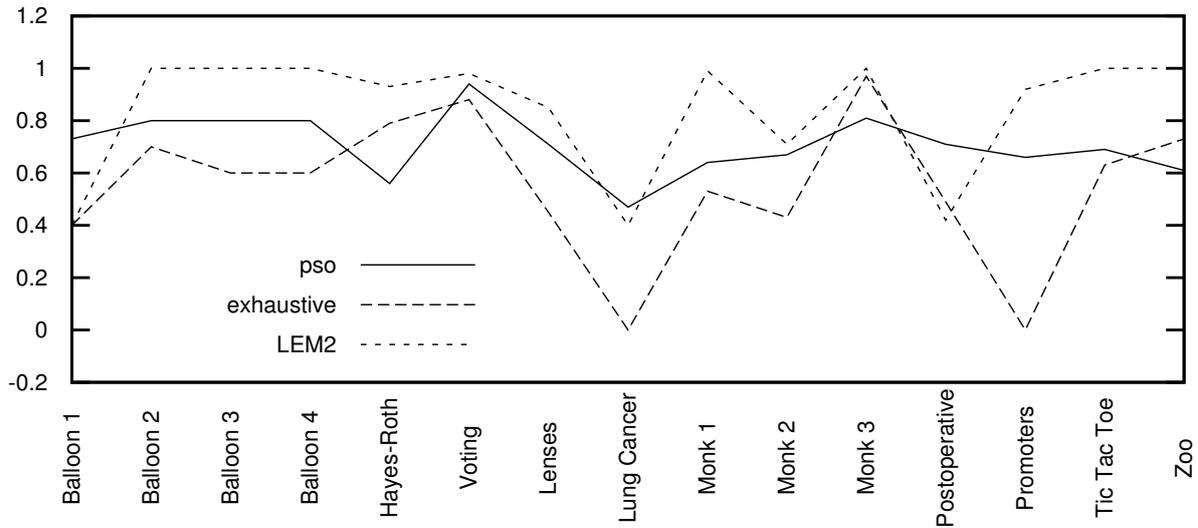


Fig. 2. Accuracies of proposed algorithm (ps), exhaustive one and LEM2

the number of attributes and y-axis shows:

- speedup for the first graph,
- efficiency for the second graph.

Although, with an increasing number of attributes speedup and efficiency are generally growing, the tendency is not clear.

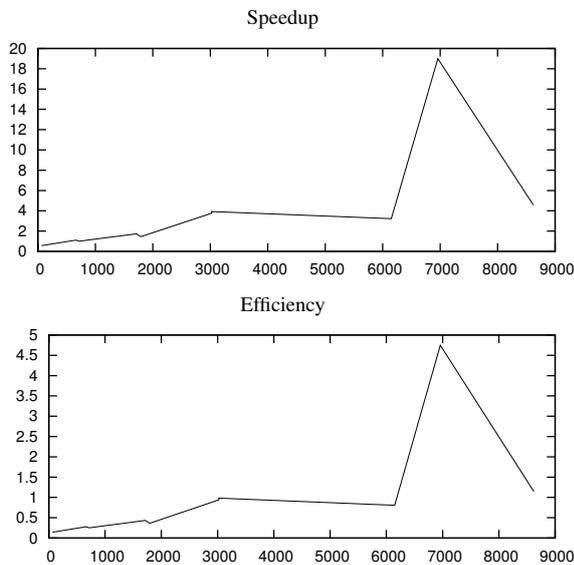


Fig. 4. Relationship between size of table and speedup or efficiency

On Figure 4 the relationship between the size of table and speedup or efficiency is shown. The horizontal axis corresponds to the table size, i.e. the number of attributes multiplied by the number of objects. Similarly to Figure 3, vertical axes correspond to speedup and efficiency. On Figure 4 it can be almost clearly seen, that, with an increasing size of data table, speedup and efficiency grow. The trend is only

disturbed by the case of the Voting table, which was already described.

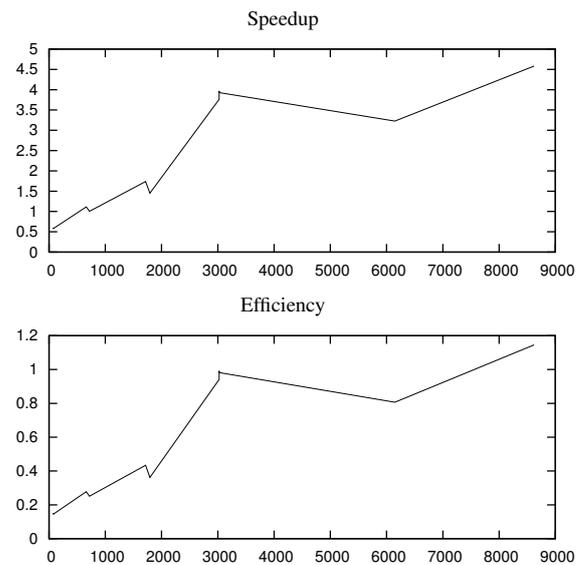


Fig. 5. Relationship between size of table and speedup or efficiency with Voting table removed

Relationship between size of table and speedup or efficiency, with removed the Voting table statistics, is shown on Figure 5. Graphs on Figure 5 clearly show upward trend in speedup and efficiency of proposed algorithm with the growth of the size of the table. When overhead of managing threads and parallelism is compensated by the table size, a parallel version of proposed algorithm performs better than the sequential one.

If there is any relation between the number of decision

classes and performance of proposed algorithm, it cannot be induced from gathered results. The main parameters, which can help to decide if a parallel or sequential version should be used, are: the number of attributes and the number of objects.

V. CONCLUSION

Although the presented algorithm was not able to achieve astonishing accuracy, it was on par with the ones implemented in RSES system used with default settings. As with all settings, they can be tuned to selected problem to achieve better results. It is noteworthy, that proposed algorithm performed better in most cases than exhaustive algorithm from RSES with rule shortening turned off and simple voting as a method for solving conflicts.

Particle swarm optimization is said to be simpler to apply in purely numerical optimization problems than genetic algorithms. It is due to its straightforward application and there is no need to define complex operators, as in genetic algorithms. Unfortunately, applying particle swarm optimization to feature selection and calculating reducts is not as straightforward as applying it to numerical optimization problems. Coding attributes subsets as binary strings and updating particles' speed and position turned out to be as complicated as when calculating operators in genetic algorithms.

The performance of parallel version of presented algorithm is promising. With growing size of data tables, algorithm's speedup and efficiency were raising. Although, these are optimistic results, there should be examined what impact on these coefficients would have adding more cores or processors.

The performance results are even more optimistic, as presented algorithm is a part of a *Data Mining EXpression Library* (dmexl). The dmexl library is a framework for easier development of data mining algorithms. Currently, efforts are taken into providing building blocks for feature selection algorithms. From the user's perspective, library enables writing complex algorithms as a simple expression. By selecting executor object, the user decides if algorithm should be executed sequentially or in parallel. Some examples can be seen in library's source code, which is available on the Internet: <https://github.com/mateka/dmexl>.

Another interesting research topic, would be using modification of proposed algorithm to execute feature selection in the context of clustering. As a classification accuracy would be unavailable, some other measure should be used to grade

obtained particles. One possibility is to use a measure of coherence of resulting clusters. If this approach is sensible and applicable, further research has to be done.

REFERENCES

- [1] Guyon, I., Elisseeff, A.: An introduction to variable and feature selection, *The Journal of Machine Learning Research*, 3, 1157–1182, JMLR. org, 2003
- [2] Fleuret, F.: Fast binary feature selection with conditional mutual information, *The Journal of Machine Learning Research*, 5, 1531–1555, JMLR. org, 2004
- [3] Das, S.: Filters, wrappers and a boosting-based hybrid for feature selection, *ICML*, 1, 74–81, Citeseer, 2001
- [4] Wang, X., Yang, J., Teng, X., Xia, W., Jensen, R.: Feature selection based on rough sets and particle swarm optimization, *Pattern Recognition Letters*, 28, 4, 459–471, Elsevier, 2007, <http://www.dx.doi.org/10.1016/j.patrec.2006.09.003>
- [5] Pawlak, Z.: Information systems theoretical foundations, *Information systems*, 6, 3, 205–218, Elsevier, 1981 [http://www.dx.doi.org/10.1016/0306-4379\(81\)90023-5](http://www.dx.doi.org/10.1016/0306-4379(81)90023-5)
- [6] Widz, S. and Slezak, D., *Rough Set Based Decision Support – Models Easy to Interpret. Selected Methods and Applications of Rough Sets in Management and Engineering*, 95-112, Peters, G., Lingras, P., Slezak, D., Yao, Y., *Advanced Information and Knowledge Processing*, Springer, 2012, http://www.dx.doi.org/10.1007/978-1-4471-2760-4_6
- [7] Komorowski, J., Pawlak, Z., Polkowski, L., Skowron, A.: *Rough sets: A tutorial, Rough fuzzy hybridization: A new trend in decision-making*, 3–98, Springer Verlag, Singapore, 1999
- [8] Stefanowski, J.: On rough set based approaches to induction of decision rules, *Rough sets in knowledge discovery*, 1, 1, 500–529, Heidelberg, Germany: Physica-Verlag, 1998
- [9] Bazan, J. G.: A comparison of dynamic and non-dynamic rough set methods for extracting laws from decision tables, *Rough sets in knowledge discovery*, 1, 321–365, Citeseer, 1998
- [10] Bazan, J. G., Szczuka, M.: RSES and RSESLib—a collection of tools for rough set computations, *Rough Sets and Current Trends in Computing*, 106–113, Springer Berlin Heidelberg, 2001, http://www.dx.doi.org/10.1007/3-540-45554-X_12
- [11] Eberhart, R., Kennedy, J.: *Particle Swarm Optimization*, *Neural Networks*, 1995., IEEE International Conference on, 1942-1948, IEEE, 1995, <http://www.dx.doi.org/10.1109/ICNN.1995.488968>
- [12] Shi, Y., Eberhart, R.: A modified particle swarm optimizer, *Evolutionary Computation Proceedings*, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on, 69–73, IEEE, 1998, <http://www.dx.doi.org/10.1109/ICEC.1998.699146>
- [13] Chu, S.-C., Roddick, J. F., Pan, J.-S.: Parallel particle swarm optimization algorithm with communication strategies, submitted to *IEEE Transactions on Evolutionary Computation*, 2003
- [14] Koh, B., George, A. D., Haftka, R. T., Fregly, B.: Parallel asynchronous particle swarm optimization, *International Journal for Numerical Methods in Engineering*, 67, 4, 578–595, Wiley Online Library, 2006, <http://www.dx.doi.org/10.1002/nme.1646>
- [15] Goldberg, D.: *Genetic algorithms in search, optimization, and machine learning*, Addison-wesley Reading Menlo Park, 1989
- [16] Bache, K., Lichman, M.: *UCI Machine Learning Repository*, <http://archive.ics.uci.edu/ml>, 2013, University of California, Irvine, School of Information and Computer Sciences