Genetic algorithms in decomposition and classification problem.

Jakub Wróblewski
Institute of Mathematics,
Warsaw University
02-097, Banacha Str. 2, Warsaw Poland
e-mail: jakubw@jakubw.pl

1 Introduction

Some combinatorical problems concerned with using rough set theory in knowledge discovery (KD) and data analysis can be successfully solved using genetic algorithms (GA) - a sophisticated, adaptive search method based on the Darwinian principle of natural selection (see [4], [6]). These problems are frequently NP-hard, as in case of reducts or templates finding (see [13]), and there is no fast and reliable way to solve them in deterministic way.

Genetic algorithms are flexible and universal - they can be used in various situations. On the other hand, approximate but fast heuristics are known for many of considered tasks. They are designed and tuned up especially for a problem, and often are more efficient than simple genetic algorithm. Unfortunately, they are often suboptimal and cannot avoid local optima. Moreover, if they are deterministic, there is no hope for improvement even if one can spend more time on computations.

The advantages of both genetic and heuristic algorithms can be exploited by hybrid algorithms - nondeterministic, problem-oriented heuristics controlled by genetic algorithm. In this work some examples of hybrid systems are presented, as well as the theory of order-based genetic algorithms being used in these systems.

In the second section of this work an idea and general scheme of hybrid algorithms is discussed in details.

In the third section a theory of order-based genetic algorithms is presented, as well as an overview of order-based genetic operators used in various situations.

The next section involves two examples of application of hybrid algorithms supporting rough set methods in knowledge discovery: a system for short reducts generation and a method of templates finding (see [13] for comparison with another methods).

In the last section ordinal-based genetic algorithms are presented. This is a new, promising technique which can be used in hybrid algorithms in some problems, when order-based GA are not appropriate. An NP-hard problem of matrix covering is presented as an example of application.

2 Hybrid algorithms

In a case of *classical genetic algorithms* (see [4], [6]) we are given a state space S (finite, but large) and a function: $f : S \to R_+$. Our goal is to find x_o: $f(x_o) = max\{f(x)\colon\ x \in S\}$. Elements of set S are "*individuals*". The main idea of classical genetic algorithms is based on the Darwinian principle of natural selection. We treat a value of the function f as ability to survive in the environment ("*fitness*"), and we simulate the process of evolution as follows:

1. We choose the representation scheme (and change "individuals" to "chromosomes" - usually bit strings).
2. We randomly choose the set of chromosomes as an initial population.
3. We calculate "fitness" $F(c)$ of each chromosome c as a value of $f(s(c))$, where $s(c)$ is the individual encoded by c. Then we create a new population, replacing the chromosomes with low fitness by those with higher fitness.
4. We randomly affect the new population by *genetic operators*, e.g. *mutation* (small, random modifications of chromosomes) and *crossing-over* (exchange of "genetic material" between some pairs of chromosomes).
5. We repeat 3-4 with the new population, until a stopping criterion is satisfied.

The result of evolution is the best individual x_{max} which is usually nearly as good as the global optimum x_o.

The scheme presented above is general and domain-independent. In a case of real application we have to answer to a few questions - some of them are crucial for efficiency of our algorithm. We have do choose a method of individuals' representation (a genetic code), a method of selection, genetic operators, a stopping criterion, other parameters (e.g. population size, probabilities of mutation and crossing-over). Theory and practice of genetic algorithms give us some suggestions how to design the genetic system [4]; detailed solutions are usually domain-dependent.

A class of *heuristic algorithms* is hard to characterise. Roughly speaking, these are approximate deterministic algorithms, often based on intuitions, using problem-dependent mechanisms and tricks. They are often based on a *greedy* paradigm: construct a result step by step, in each of them achieve as much as

possible. We usually get a good, but not necessarily optimal solution, depending on an initial state or parameters of algorithm.

A nice example of using a kind of heuristic algorithm in an NP-hard problem is one described in [3]. A method of solving the graph coloring problem, which is one of the classical NP-hard problems [5], is presented there.

A version of the graph coloring problem considered in [3] can be formulated as follows: given a graph with weights assigned to nodes and n colors, assign colors to nodes in such way that no adjacent nodes have the same color; if it is not possible to color all the graph, maximize the sum of weights of colored nodes. A greedy algorithm for this problem acts as follows: sort nodes in ascending order using weights, then for each node assign the lowest possible color number. This strategy produces good, but often suboptimal solutions. As the next step, an order-based genetic algorithm was used to produce a permutation of nodes. This permutation was used as control sequence in the greedy algorithm: now nodes are considered in the order generated by this permutation. Results obtained by this hybrid algorithm were significantly better than these obtained by the greedy algorithm as well as by the greedy algorithm controlled by random permutations.

The hybridisation strategy described in [3] represents more general approach to combinatorical optimisation problems. The general scheme of *hybrid algorithm* is as follows:

1. Find a strategy (heuristic algorithm) which gives an approximate result.
2. Modify (parametrise) the strategy using a control sequence, so that the result depends on this sequence (recipe).
3. Encode the control sequence to a chromosome.
4. Use a genetic algorithm to produce control sequences. Proceed with the heuristic algorithm controlled by the sequence. Evaluate an object generated by the algorithm and use its quality measure as a fitness of the control sequence.
5. A result of evolution is the best control sequence, i.e. the sequence producing the best object. Send this object to the output of the hybrid algorithm.

When a new hybrid algorithm is designed, we should remember about two following conditions:

1. All control sequences generated by our genetic algorithm should be properly recognised and used by a heuristic algorithm.
2. All potential solutions should be accessible using our algorithm, i.e. there should exist a control sequence which produces it.

The first condition can be omitted: if the heuristic algorithm encounters an unintelligible control sequence, it simply skips it and assigns zero as a fitness value. The second condition is necessary.

A well-constructed hybrid algorithm has many advantages over both a simple genetic algorithm (with literal encoding of search space), and heuristic strategies:

- Since control sequences are generated by a nondeterministic genetic algorithm, the more time we spend on computations, the better result we obtain. There is no such an advantage in case of deterministic heuristics: we get just one suboptimal solution.
- A genetic algorithm produces a number of different suboptimal solutions of good quality.
- A simple genetic algorithm (or another search technique) with literal encoding of state space evaluates many very bad solutions. A hybrid algorithm actually searches only a part of state space - a class of good, but not optimal, solutions. Every evaluation (via a heuristic algorithm) generates quite good solution. Therefore we can terminate evolution process even after the first step, and we will get a good solution; this feature can be important in some real-time applications.
- Because of its modular structure, hybrid algorithms should be easy designed and reused in other, similar applications. As on Figure 1, we can use one, universal genetic library to support many heuristic algorithms. An example of a situation, when one order-based genetic algorithm controls two different tasks in knowledge discovery system, is described in one of the next sections.

An order-based genetic algorithm is one of the most widely used component of various hybrid systems. Theoretical foundations and practical construction of this algorithm are presented in the next section. Another type of genetic algorithm, an ordinal one, can be easily used as a generator of control sequences. This promising technique is described in the last section of this work.

3 Order-based genetic algorithms

3.1 An overview of algorithm

In the *order-based genetic algorithms* a chromosome is an n-element permutation σ, represented by a sequence of numbers: $\sigma(1)\ \sigma(2)\ \sigma(3)\ \ldots\ \sigma(\mathrm{n})$. The evolution proceeds as follows:

1. Initial population of M individuals is generated randomly.

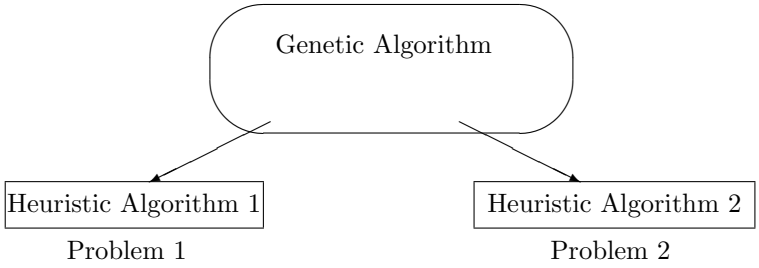

Figure 1: A hybrid system.

2. The population is affected by genetic operators of mutation and recombination.

3. Fitness function of every individual is calculated. In the case of a hybrid algorithm a heuristic part is launched under control of individual; a fitness value depends on the result of heuristic algorithm. See the next section for details.

4. New population is generated using "roulette wheel" algorithm: the fitness value of every individual is normalised and treated as probability distribution on population; then randomly choose M new individuals using this distribution.

5. Repeat from step 2. Stop after obtaining good results.

Mutation of order-based individual means one random transposition of its genes:

$$\mathbf{1}\ \ \mathbf{2}\ \ 3\ \ 6\ \ 5\ \ 4 \overset{Mutation}{\longrightarrow} \mathbf{2}\ \ \mathbf{1}\ \ 3\ \ 6\ \ 5\ \ 4$$

There are various methods of recombination (*crossing-over*) considered in literature. In [4] such methods as PMX (Partially Matched Crossover), CX (Cycle Crossover) and OX (Order Crossover) are described.

The PMX operator was used in the Traveling Salesman Problem [4], where a state space of possible paths of a salesman through a set of cities was literally encoded in the form of permutations (an order of visiting the cities). In the PMX method a matching section is set for both parents as a part of chromosome with a random beginning and end. Then, the adjacent genes from matching sections of both parents are combined into pairs. Finally, these pairs are treated as transpositions; both parents are altered due to these transpositions. E.g.:

$$\begin{array}{ccc|cc|c} 1 & 2 & 3 & 4 & 5 & 6 \\ 2 & 1 & 4 & 6 & 3 & 5 \end{array} \overset{PMX}{\longrightarrow} \begin{array}{cccccc} 1 & 2 & 5 & 6 & 3 & 4 \\ 2 & 1 & 6 & 4 & 5 & 3 \end{array}$$

Vertical lines indicate matching section.

Other type of crossover operator, UOX (Uniform Order-based Crossover) was used in the hybrid algorithm described in [3].

We will use another type of crossing-over operator for further analysis: MOX (Modified Order Crossover), [23]. This recombination operator affects two parent chromosomes and replaces them by two children. First, we choose one gene in first parent's chromosome at random. This gene will be the end of a matching section, starting at the beginning of chromosome. Identical matching section is marked on second parent's chromosome. Then, we leave the matching sections unchanged, but the rest of genes of the first chromosome is set in the order of appearance in the second chromosome. We perform the same operation on the second parent. For example:

$$\begin{array}{ccc|ccc|c} 1 & 2 & 3 & 4 & 5 & 6 & \\ 4 & 2 & 1 & 3 & 6 & 5 & \end{array} \overset{MOX}{\longrightarrow} \begin{array}{ccc|ccc} 1 & 2 & 3 & 4 & 6 & 5 \\ 4 & 2 & 1 & 3 & 5 & 6 \end{array}$$

Vertical line indicates the end of matching section.

3.2 Theoretical foundations

One of the most important results concerned with a theoretical foundations of classical genetic algorithm is a notion of schema and schema theorem [6], [4]. This approach is focused on *schemata* - subsets of the state space described by a certain combination of genes' values. Due to this theory, an efficiency of genetic algorithms is concerned with the "implicit parallelism": the algorithm favores better (in average) schemata; each individual is a realisation of many different schemata, so the algorithm works with much more schemata than individuals. One must believe, that the optimal solution is possible to reach by combining "good" schemata; this approach is called "building block hypotesis". The *schema theorem* formulates an estimation of a number of schema representatives after one step of evolution - this estimation shows, that the average number of good (in a sense of average fitness) schemata will grow expotentially.

The notion of schema can be generalised into the case of order-based genetic algorithms. Some of these generalisations are described in [4]. The notion of schema (or *o-schema*, for "order-based") can be defined in different ways, but useful definitions should obey some principles (formulated in [18]) binding schemata and operators of crossing-over. On the other hand, a reasonable schema definition should refer to an internal structure of state space. In the case of hybrid algorithms, a notion of *rd-schemata* (relative dispersed) is shown [23] to be more adequate than those presented in [4].

Definition: Relative dispersed schema $\mathrm{rd}^n(\ \mathrm{a}_1\ \mathrm{a}_2\ \ldots\ \mathrm{a}_k\ )$ expands to the set of all individuals of length n with genes $\mathrm{a}_1\ \ldots\ \mathrm{a}_k$ located in this order (not necessarily sequentially). The *order* $o(\cdot)$ of an rd-schema is defined to be equal to the number k.

This kind of o-schemata was used in [3] and described in details in [23]. The modified order crossover (MOX) operator is concerned with the rd-schemata. An analogy of the classical schema theorem can be formulated for a given relative dispersed schemata S and MOX operator:

$$E\left(N_{t+1}\right) \geq N_t \frac{f}{\overline{f}} \left(1 - p_{cross} \cdot Pc \cdot \left(1 - \frac{N_t}{M}\right) - p_{mut} \cdot Pm\right)$$

where:
N_t - a number of representatives of S in the t-th population,
f - an average value of fitness function in population,
$\overline{f}$ - an average value of fitness function of individuals matching S,
p_{cross} - a probability of crossing-over,
p_{mut} - a probability of mutation of individual,
M - a number of individuals in population,
Pm - a probability of disruption of schema S by a mutation,
Pc - a probability of disruption of schema S by the MOX operation.
The values of Pc and Pm was calculated in [23] as:

$$Pc \leq \frac{1}{n+1} \sum_{k=0}^{n} \frac{\left(\begin{array}{c} n \\ l \end{array}\right) - \left(\begin{array}{c} k \\ l \end{array}\right)}{\left(\begin{array}{c} n \\ l \end{array}\right)} \cdot \frac{\left(\begin{array}{c} n \\ l \end{array}\right) - \left(\begin{array}{c} n-k \\ l \end{array}\right)}{\left(\begin{array}{c} n \\ l \end{array}\right)}$$

$$Pm = \frac{l}{n^2 - n} \left(l - 1 + \frac{2 \cdot (n-l)}{n \cdot (n-1)} \sum_{k=1}^{n-2} 2\,(n-k-1)\, \frac{\left(\begin{array}{c} n-2 \\ k \end{array}\right) - \left(\begin{array}{c} n-l-1 \\ k \end{array}\right)}{\left(\begin{array}{c} n-2 \\ k \end{array}\right)} \right)$$

where:
n - a number of genes in chromosome,
l - an order $o\left(S\right)$ of scheme S.

The main result is nearly identical with the classical case. Therefore we may use classical results like the building block hypothesis, the analogy with k-armed bandit, theory of deceptiveness in the order-based genetic algorithms.

There are other approaches to the problem of genetic algorithms convergence and efficiency. Many of them are based on Markov chain as a tool for modelling behaviour of GA. Some interesting results are presented in [16], [21].

In [16] authors model a simple genetic algorithm using a Markov chain with N states - each of it corresponds to one possible population. In case of order-based GA, number N can be calculated by:

$$N = \left(\begin{array}{c} M + n! - 1 \\ n! - 1 \end{array} \right)$$

where M is the number of individuals in population, n - size of chromosome (permutation). The size of transition matrix Q of the chain is $N \times N$, so it is too large to perform direct analysis.

Some of the results concerned with the classical case are recalculated in [23] for the order-based genetic algorithms. Particularly, the Markov chain describing order-based GA was shown to be ergodic (as in [16]) and the convergence rate was estimated (as in [21]). Moreover, an optimal mutation probability was calculated (unfortunately, this result is practically useless, because it assumes complete knowledge about the fitness function, including its extrema etc.).

Results presented in [23] suggests, that the order-based genetic algorithms used in hybrid systems are based on as reliable theoretical foundations as the classical GAs are. On the other hand, these foundations are still not satisfactory from the practical point of view - in both classical and order-based case.

4 Applications

4.1 Short reduct finding using genetic algorithm

The notion of a reduct is one of the main notions of rough set theory [17].

Suppose we have an *information system* (see [13]): a set of m object $\{o_1, ...o_m\} = U$, each described by n attributes $\{a_1, ...a_n\} = A$. We have a decision value d assigned to each object. Suppose that every two objects with different decision values have different values of at least one attribute.

Definition: Let a *(relative) reduct* $R \subseteq A$ of an information system be a minimal (with respect to inclusion) subset of attributes satisfying the following property:

$$\forall k, l \; \forall a_i \in R : a_i(o_k) = a_i(o_l) \Longrightarrow d(o_k) = d(o_l)$$

The problem of finding a globally minimal reduct for a given information system is NP-hard (see [20]). On the other hand, short reducts are often used to build effective decision algorithms: if a subset of attributes is known as a reduct, we are able to predict $d(x)$ using only this subset of data. The decision rules generated from this subset should be (if the reduct is short enough) simple and easy to find.

4.1.1 Classical genetic algorithm

One can use classical, binary genetic algorithm [22] to generate reducts: every binary individual encodes one subset of attributes - a potential reduct. For

example:

$$1001001100 \longleftrightarrow \{a_1, a_4, a_7, a_8\}$$

A classical binary mutation and crossover operators [4] and the "roulette wheel" selection algorithm are used. The fitness function of a subset R has a form:

$$F(R) = \frac{n - L_R}{n} + \frac{2C_R}{m^2 - m}$$

where L_R denotes a number of "1" in the subset R, and C_R denotes the number of object pairs (with different decision values) discerned by the attribute subset R.

The process of calculating C_R is the most time-consuming operation; to make it faster an additional structure called a *distinction table* is generated for an information system. This is a binary matrix of size $(n+1) \times \left(m^2 - m\right)/2$. Each column of the matrix corresponds to one attribute, each row corresponds to one pair of different objects. Value "1" means, that an attribute has different value on the pair of objects. To find a reduct means to find a column covering of the matrix.

The results of this experiment are described in [22]. However, these results were still not satisfactory, because there was no sure, that an obtained subset is a reduct (not a superreduct), and the second method of generating of short reducts was introduced: a hybrid algorithm.

4.1.2 Hybrid algorithm

In the hybrid algorithm [22] a simple, deterministic method was used for reduct generation:

1. Let R be a set of all attributes and let $(b_1 \ldots b_n) = \tau(a_1 \ldots a_n)$ be an ordered list of attributes - the order is represented by a permutation τ.
2. For $i = 1$ to n repeat steps 3 and 4:
3. Let $R \longleftarrow R - b_i$.
4. If R does not satisfies a condition from the definition of reduct, undo step 3.

The result of the algorithm will always be a reduct. Every reduct can be found using this algorithm, the result depends on the order of attributes (proof: see [22]). The genetic algorithm is used to generate the proper order. To calculate the function of fitness for a given permutation (order of attributes) we have to perform one run of the deterministic algorithm and calculate the length of the reduct found. The function of fitness depends only on this length:

$$F(\tau) = \frac{1}{L_\tau}$$

Another fitness formula was considered also in [15]:

$$F(\tau) = n - L_\tau + 1$$

and the results improved. Linear scaling [4] was used in both cases.

The hybrid algorithm described above performs much slower that the classical one. On the other hand, the reducts obtained by this algorithm are usually shorter. Moreover, the hybrid algorithm generates from 50 to 500 different reducts in comparison with 5 to 50 reducts generated by the classical GA at the same time.

In [15] an improvement of the hybrid algorithm is presented. General scheme is the same as described above, but a procedure of testing whether a subset is a reduct, was improved. Now the algorithm acts as follows:

1. We have to determine, whether a subset s is a reduct or superreduct (a superset of a reduct). First, we search a list of known reducts. If there exists a reduct r such that $r \subset s$, the answer is YES.

2. We search for s in a treelike structure of known subreducts. If we find s, the answer is NO.

3. If both of previous steps fail, we are looking in our data for a pair of objects with equal values on attributes from s, but with the different decisions. We are inserting all objects from the table into a binary tree - this is equivalent to sorting the data by the attribute values on s. Now the pair is easy to find - if one exists. If such a pair is found, we can reorder objects and bring the pair to the beginning (it can save up to 25% of time).

4. If we find such a pair, the answer is NO and we can add s to the tree of subreducts.

5. If there is no such a pair, the answer is YES and we can add s to the list of (super)reducts, removing all its supersets.

This new procedure is not only faster, than a method based on the distinction table (see above), but also occupies much less space. The results of computation time for some data tables are presented in Figure 2. A set of up to 30 short reducts was generated in each case. All computations were performed on a Pentium-100 machine.

In [23] the average efficiency (the average length of obtained reducts) using three recombination operators: PMX, OX and MOX is compared. The results

Table size obj × attr	Comp. time sec
4,495 × 37	63
30,000 × 10	61
471 × 33	6.2
225 × 490	69
15,534 × 16	3

Figure 2: Computation time for various data sets.

obtained using MOX were better than those obtained using OX, and were significantly better than those obtained using PMX operator (based on absolute positions of attributes). This suggests, that rd-schemata (concerned with the MOX operator) create a good tool for analysing this hybrid algorithm.

4.1.3 Optimisation based on the number of rules

Both algorithms described above generate possibly shortest reducts, i.e. the reducts with as few attributes as possible. On the other hand, our goal is not to calculate reducts, but to construct an efficient system for classification or decision making. The set of decision rules generated from the set of short reducts should be general enough to deal with a new object. This assumption creates a foundation of decision systems described in [1], [10], [12], [15].

Another approach is to select a reduct due to the number of rules it generates rather than to its length. Every reduct generates an indiscernibility relation (see [13]) on the universe and in most cases it identifies some pairs of objects. Therefore the number of rules generated by a reduct is less than the number of objects. If a reduct generates less rules, it means, that the rules are more general and they should better recognise new objects.

The number of rules can be easily computed due to the improvement of the reduct generation system described in [15] (see pervious section). When the program determines whether a subset is a reduct, it constructs a tree of all objects. The number of nodes in this tree is equal to the number of rules generated by the reduct. So the number of rules can be computed with nearly no additional time. The hybrid algorithm described in the pervious section can be used to find reducts generating the minimal number of rules. All we have to do is to change the fitness formula:

$$F(\tau) = m - R_\tau + \frac{n - L_\tau + 1}{n}$$

where R_τ denotes the number of rules generated by the reduct. Now the primary criterion of optimisation is the number of rules, the secondary is the

m_1	m_2	n	Time sec.	Minimal no. of rules	Minimal red. length	R
357	154	41	18	44.1%	33.1%	5.7%
24000	6000	10	52	97.1%	97.1%	2.3%
4435	2000	37	284	27.1%	26.7%	0.1%
800	200	81	397	76.4%	76.6%	5.0%
16000	4000	17	138	44.2%	43.3%	0.6%

Figure 3: Results of classification.

reduct length.

Some data sets were used in experiments to compare the classification efficiency of systems basing on the shortest reducts and the smallest sets of rules. The results are presented on Figure 3, where m_1 is a number of objects in training set, m_2 - a number of objects in test set, n - a number of attributes, R - a reduction rate of rule set. The results involve a training time (Pentium-100) and a number of correct classifications of test objects, basing on rule set generated by the best 30 reducts found using either the old algorithm or the new one.

We can conclude, that the classification system based on the reducts optimised due to the number of rules performs better (or not worse) than the short reduct based one. Moreover, due to the rule set reduction, it occupies less memory and classifies new objects faster.

4.2 Finding templates using genetic algorithm

In the article [13] in this book the problem of template generation is presented in details, including some heuristic methods of template generation. We are interested in good templates, i.e. templates with many fixed positions and matching many objects. Some versions of the problem of finding the best (in this sense) template is proven to be NP-hard. Another method for template generation, a hybrid algorithm (as in [11], [15]), is presented below.

Note that every template can be represented by a binary string of length n (indicating which attributes are fixed) and any object matching it (called a base object). This representation will be the most natural in problems as follows: "Find the largest template matching the object x_0". There are also other questions, like: "Find the globally best template in this information system", that can be reformulated in terms of finding the best template matching one specific object. In fact, the heuristic algorithm described below finds a good template for a given base object:

1. Get an object x_0 as a base object.
2. Let σ be an order of attributes.

3. Consider a set of templates of the form: $T_1 = (a_{\sigma_1} = v_{\sigma_1})$, $T_2 = (a_{\sigma_1} = v_{\sigma_1}) \wedge (a_{\sigma_2} = v_{\sigma_2})$, etc., where v_i denotes a value of i-th attribute on x_0.

4. Choose the best template among T_1,...,T_n. This is a result generated by permutation σ.

Every locally maximal template can be found using this algorithm - the result depends on the order of attributes. The time complexity of this algorithm is bounded by $O(n^2m)$. The algorithm described above can be optimised in many ways: for example, objects that do not match the base object in any position can be deleted from database (a pre-processing). Our goal is to find the proper order of attributes and we use genetic algorithms to do this job. Our chromosome will be a permutation of length n, and its fitness value depends on a quality of a template T found by the heuristic algorithm.

The value of the fitness function, $F(T)$, depends on two parameters viz. the number O_T of objects in the data table matching the given template (base object excluded) and the number A_T of fixed positions:

$$F(T) = O_T \cdot A_T$$

Calculating $F(T)$ is the most time-consuming operation, but some techniques can be used to make it faster. For example, we can store fitness values in memory and try to recall them instead of calculating them again.

The "roulette wheel" algorithm was used as a selection strategy. The results were slightly better in the case when the elitist strategy was used as an additional technique. The best individual was copied to the new population without changes.

Crossing-over affects chromosomes selected to reproduction with the probability of $P_c = 0.7$. We use PMX [4] and MOX [23] method.

There is no simple and fast way to generate globally good templates by this algorithm - generating the best template for any object is rather unacceptable for large databases. On the other hand, a globally good template means that it matches many objects from the database (in examples presented below - up to 60%). Note, that if we choose any of them as our base object, we find global optimum. So, we can simply choose randomly as many objects as possible and calculate the templates - the largest of them will probably be a global optimum.

See [13] for experimental results in comparison with other algorithms.

Note, that the genetic algorithm used in the hybrid system described above is identical (except of fitness function) with that used to produce short reducts (see previous section). Actually, in our real knowledge discovery system [8] with both these hybrid algorithms implemented, control sequences generates one genetic algorithm.

The same hybrid algorithm can be used to produce decision templates. We have only to change the fitness formula:

$$F(T) = c_1 \left(\frac{O_+}{O_+ + O_-} \right)^3 + c_2 \left(\frac{O_+}{O_+ + O_{used}} \right)^3 + \frac{O_+}{O_{class}}$$

where:

O_+ = the number of objects matching the template and belonging to the decision class;

O_- = the number of objects matching the template and not belonging to the decision class;

O_{used} = the number of objects matching the template and belonging to the decision class, but already covered by the previously generated templates.

O_{class} = the number of objects belonging to the decision class;

c_1, c_2 - parameters.

Using this fitness function, we can cover whole decision class with templates. We start with a random object in the class as a base object, then the obtained template is stored in the memory. The next base object is chosen randomly from the class, but we do not take into account objects covered by the already produced templates.

Observe that a high value of c_1 forces the templates chosen not to cover any object outside the class, but the templates are shorter and we have to use more of them. If the value of c_2 is high then obtained templates are separated. We can use the conjunction of obtained templates as our approximate definition of the decision class.

The results of experiments presented in [11] show, that in real cases we have to be very careful in choosing value of c_1. If the value is too high, then we obtain too many templates; if the value is too low, the obtained decision algorithm is useless: it cannot recognise properly too many objects. On the other hand, there are many templates covering only 1 or 2 objects. These "exceptions" can be omitted if we are interested in short and general rules.

5 Ordinal-based genetic algorithms

The flexibility and universality of order-based genetic algorithms cannot assure efficiency of all possible hybrid applications. Thus, there is a need for other types of genetic algorithms able to control various heuristic procedures. An example of such a genetic algorithm is an ordinal-based one.

In the *ordinal-based genetic algorithm* an individual has the following form:

$$\left\{ \begin{array}{ccccccc} x_1 & x_2 & x_3 & x_4 & \cdots & x_n \end{array} \right\}$$

where n is a chromosome length, $x_i \in [1, ..., b]$ and b is a given (usually small) integer.

Mutation operator affects a single random gene in individual increasing it (with probability P_+) or decreasing by one. E.g.:

$$\{\ 1\quad 1\quad \mathbf{2}\quad 1\quad 3\ \} \overset{Mutation}{\longrightarrow} \{\ 1\quad 1\quad \mathbf{3}\quad 1\quad 3\ \}$$

Decreasing of value 1 is ignored.

Crossing-over operator is very similar to the classical one [4]. A matching section is chosen randomly, then the values of genes in matching sections of parents are exchanged. E.g.:

$$\begin{array}{ccc|cc} 1 & 2 & 1 & 1 & 4 \\ 2 & 3 & 1 & 2 & 2 \end{array} \longrightarrow \begin{array}{ccc|cc} 2 & 3 & 1 & 1 & 4 \\ 1 & 2 & 1 & 2 & 2 \end{array}$$

Vertical line indicates the end of matching section.

This type of genetic algorithm can be used in hybrid system solving the matrix covering problem [2], [5]. We have a binary matrix of size $n \times n$. Our goal is to find a covering, i.e. a subset of columns, such that in each row there exists at least one "1" in one of the columns from the covering (if possible). The problem of finding the smallest covering is one of the classical NP-hard problems. On the other hand, some NP-hard problems concerned with the rough set theory (e.g. short reducts finding) can be formulated in terms of minimal covering of a matrix.

Let us consider a simple "greedy" heuristic for this problem:

1. Find a column covering the most rows in the matrix.
2. Choose this column to our covering. Remove covered rows from matrix.
3. Repeat from step 1 until the matrix is covered.

This algorithm gives good, but suboptimal solutions. We will transform it to a hybrid version:

1. Let $i = 1$. Let $\{x_1, ...x_n\}$ be an ordinal-based individual.
2. Sort columns in descending order with respect to the number of covered rows.
3. Choose x_i-th column for our covering. Remove the covered rows from matrix.
4. Let $i = i + 1$.
5. Repeat from step 2 until the matrix is covered.

For example, if our individual has a form $\{1, 2, 1, ...\}$, we choose the best column in the first step, the second column in the second step, the best column in the third step etc... Now the algorithm can find every possible covering, including the optimal one. The formula of fitness function depends on the size of covering found.

Experiments performed on the special case of the matrix covering problem used in cryptography show, that the hybrid algorithm controlled by the ordinal-based genetic algorithm acts better than another methods.

6 Conclusions

A hybrid algorithm is a system where a genetic algorithm controls behaviour of a simple heuristic search method. These algorithms can be successfully used in NP-hard problems related to the rough set methods in KD and data mining. Application of hybrid methods in such problems like short reducts finding or template finding was described in this work.

There are two methods of short reducts finding presented in this work: a classical genetic algorithm with binary chromosomes, and a hybrid system based on the order-based genetic algorithm. Both these methods produce good results in relatively short time. The classical method is faster, but the hybrid one gives more different reducts in one pass. Moreover, the reducts obtained by hybrid algorithm are shorter; the classical method sometimes produces superreducts.

The hybrid algorithm can be easily modified to produce reducts generating the smallest number of rules. These reducts are shown to be better in classification algorithms than the shortest ones.

The order-based genetic algorithm is used also in the hybrid system which generates large templates. There are some heuristic algorithms to large templates finding as good as a hybrid system is. On the other hand, the hybrid algorithm described in this work can be easily adopted to a problem of decision templates finding, where other methods fail.

An order-based genetic algorithm was used as the driving force of hybrid systems described in this work. An overview of order-based genetic operators and the theoretical foundations was presented in section 3.

An ordinal-based genetic algorithm presented in section 5 is a new, promising technique, which can be used in hybrid algorithms. A method of usage the ordinal-based GA in rough set methods in KD, as well as its theoretical foundations, needs more researches.

Acknowledgements This work has been supported by the grant #8T11C01011 from Polish National Committee for Scientific Research (Komitet Badań Naukowych).

References

[1] Bazan J., Skowron A., Synak P. Dynamic reducts as a tool for extracting laws from decision tables, *Proc. of The Eighth International Symposium on Methodologies for Intelligent Systems* (ISMIS'94).

[2] Cormen T. H., Leiserson C. E., Rivest R. L., 1992. Introduction to Algorithms. The MIT Press, Cambridge, Massachusetts, London, England, pp.974-978.

[3] Davis L.(red.), 1991. Handbook of Genetic Algorithms. New York, Van Nostrand Reinhold.

[4] Goldberg D. E., 1989. GA in Search, Optimisation, and Machine Learning. Addison-Wesley.

[5] Garey M.R., Johnson D.S. Computers and Intractability. A Guide to the Theory of NP-Completeness. W.H. Freeman and Company New York 1979, pp.196.

[6] Holland J.H., 1992. Adaptation in Natural and Artificial Systems. The MIT Press, Cambridge.

[7] Koza J.R., 1992. Genetic Programming: On the Programming of Computers by Means of the Natural Selection. The MIT Press.

[8] Komorowski J., Ohrn A., 1997. ROSETTA - A Rough Set Toolkit for Analysis of Data. *Proceedings of the Joint Conference of Information Sciences*, March 1-5, 1997, North Carolina, USA, vol. 3, p. 403.

[9] Michalewicz Z., 1994. Genetic Algorithms + Data Structures = Evolution Programs. Springer-Verlag.

[10] Nguyen S. H., Nguyen T.T., Skowron A., Synak P., Knowledge Discovery by Rough Set Methods, *Proc. of The International Conference On Information Systems Analysis and Synthesis,* July, 22-26, 1996, Orlando, USA, pp.26-33.

[11] Nguyen S. H., Polkowski L., Skowron A., Synak P., Wróblewski J.,1996 Searching for Approximate Description of Decision Classes, *Proc.of The Fourth International Workshop on Rough Sets,Fuzzy Sets,and Machine Discovery,* November 6-8, 1996, Tokyo, Japan, pp.153-161.

[12] Nguyen S. H., Skowron A., Synak P., Rough Sets in Data Mining: Approximate Description of Decision Classes. *Proc. of The fourth European Congress on Intel-ligent Techniques and Soft Computing*, Aachen, Germany, September 2-5, 1996, pp.149-153.

[13] Hguyen S.H., Skowron A., Synak P., 1997. Discovery of Data Patterns with Applications to Decomposition and Classification Problem. Published in this book.

[14] Nguyen S.H., Nguyen H.S.,1996. Some Efficient Algorithms for Rough Set Methods. *Proc. of the IPMU-96*, Granada, Espana, pp. 1451-1456.

[15] Nguyen S. H., Skowron A., Synak P., Wróblewski J., 1997. Knowledge Discovery in Databases: Rough Set Approach. *Proc. of The Seventh International Fuzzy Systems Association World Congress*, vol. II, pp. 204-209, IFSA97, Prague, Czech Republic.

[16] Nix A., Vose M. D., 1991. Modelling Genetic Algorithms with Markov Chains. *Annals of Mathematics and Artificial Intelligence.*

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

[18] Radcliffe N. J., 1991. Forma Analysis and Random Respectful Recombination. *Proceedings of the Fourth International Conference on Genetic Algorithms.* Morgan Kaufmann Publishers, pp: 222 - 229.

[19] Skowron A., Polkowski L., 1995. Rough Mereological Foundations for Analysis, Synthesis, Design and Control in Distributive System, *Proc. Second Annual Joint Conference on Information Sciences*, Sept. 28 - Oct. 1, 1995, Wrightsville Beach, NC, pp. 346-349.

[20] Skowron A., Rauszer C., 1992. The Discernibility Matrices and Functions in Information Systems. In: R. Slowiński (ed.): *Intelligent Decision Support. Handbook of Applications and Advances of the Rough Sets Theory*, pp.331 - 362, Kluwer, Dordrecht.

[21] Suzuki J., 1993. A Markov Chain Analysis on A Genetic Algorithm. *Proceedings of the Fifth International Conference on Genetic Algorithms.* Morgan Kaufmann Publishers, pp: 146 - 153.

[22] Wróblewski J., 1995. Finding Minimal Reducts using Genetic Algorithms. *Proc. of the Second Annual Join Conference on Information Sciences*, September 28-October 1, 1995, Wrightsville Beach, NC, pp.186-189.

[23] Wróblewski J., 1996. Theoretical Foundations of Order-Based Genetic Algorithms. *Fundamenta Informaticae*, vol. 28 (3, 4), pp: 423-430. IOS Press, 1996.