Complexity-driven Evolution of Decision Graphs for Classification of Medical Data

Vili Podgorelec
Institute of Informatics, University of Maribor, FERI,
Smetanova 17, SI-2000 Maribor,
Slovenia
vili.podgorelec@uni-mb.si

Keywords: data mining, classification, meta agents, automatic programming, complexity, medical data

Received: October 27, 2004

In the paper we study the possibility of constructing decision graphs with the help of several meta agents. Decision graphs are an extension of the well known decision trees and introduce the possibility of program nodes and cycles in a classification model. A two-leveled evolutionary algorithm for the induction of decision graphs is presented and the principle of classification based on the decision graphs is described. Several agents are used to construct the decision graphs; they are constructed and evolved with the help of automatic programming and evaluated with a universal complexity measure. The developed model is applied to a medical dataset for the classification of patients with mitral valve prolapse syndrome.

Povzetek: Obravnavana je konstrukcija odločitvenih grafov s pomočjo metaagentov in njihova uporaba za klasifikacijo medicinskih podatkov.

1 Introduction

Making the right decision is becoming the key factor for the successful achievement of our goals in all areas of our work. The ways of finding the right decision are as many as the number of people who have to make them. Nevertheless, the basic idea is the same for many of them: a decision is usually made as a combination of experiences from solving similar cases, the results of recent researches and personal judgment. The number of solved cases and new researches is increasing rapidly. It could be expected that newly made decisions will become better and more reliable but for the individuals and groups who have to make decisions it is actually becoming more and more complicated, because they simply can not process the huge amounts of data anymore. And there the need for a good decision support technique arises. It should be able to process those huge amounts of data and to help experts to make their decisions easier and more reliably. For this purpose it is equally or even more important as suggesting the possible decision, to provide also an explanation of how and why the suggested decision was chosen. In this manner an expert can decide whether the suggested solution is appropriate or not.

As in many other areas, decisions play an important role also in medicine, especially in medical diagnostic processes. Decision support systems helping physicians are becoming a very important part in medical decision making, particularly in those situations where decision must be made effectively and reliably. Since conceptual simple decision making models with the possibility of automatic learning should be considered for performing such tasks, decision trees are a very suitable candidate. They have been already successfully used for many decision making purposes [Pod02].

1.1 Scope and contributions of the paper

The paper will introduce the concept of complexity-driven evolution of meta agents in classification. First the related research will be presented and the most important basic concepts defined. Then the paper will focus on agents learning with genetic programming. The concept of decision graphs will be introduced, which are an extension of decision trees. The process of agent evolution is presented with the emphasis on the complexity-based fitness function that is used to evaluate individual solutions. In the final section the application of our approach to the problem of mitral valve prolapse is shown. In the concluding section the advantages and the drawbacks of the method are presented and some future plans outlined.

The main contributions of the paper are:

- agent-based construction of decision graphs, and
- complexity-driven evolution of meta agents.

2 Related research

In today's world there is a pressing need to automate common administrative tasks in order to lower costs, minimize time spent and increase productivity. To help accommodate far-reaching lifestyle changes, new tools are needed to support imperatives of this rapidly changing environment - intelligent agents. Intelligent

agents are software programs that have the ability to act autonomously on their user's behalf, learn from experience and collaborate with other agents to achieve a common goal [Woo95]. They help their users with routine computer tasks, while still accommodating individual habits.

2.1 Intelligent agents

Intelligent agents are software programs that can identify repetitive patterns of behavior, similarities between events or things, and changes in patterns over time.

A formal specification of agents must include the representation of the following aspects:

- knowledge – the beliefs that an agent has (the information about the environment),
- engine – the actions that an agent performs and the effects of these actions,
- adapters – "ears, eyes and hands" of an agent that allow it to communicate with the environment and also with other agents over time and take the appropriate actions, and
- the goals that an agent will try to achieve.

The salient feature of agents is their adaptability and capacity for learning otherwise they are just ordinary programs. Therefore we can define an intelligent agent as an entity that owns the following properties [Woo95]:

- *autonomy*: agents encapsulate some state, and make decisions on the basis of that state without the direct human intervention;
- *reactivity*: agents are situated in certain environment, and are able to perceive changes in that environment (through the implemented sensors). They are also able to respond to changes in timely fashion;
- *pro-activity*: agents do not simply act in response to their environment, they are able to exhibit goal-directed behavior by taking the initiative;
- *social ability*: agents interact with other agents (and possibly humans) via some kind of agent-communication language, and typically have the ability to engage in social activities (such as cooperative problem solving or negotiation) in order to achieve their goals.

Agents are usually developed to provide expertise in a specific area and can, through cooperative work, jointly accomplish larger and more complex tasks. Autonomous agents have the ability to handle user-defined tasks independent of the user and often without the user's guidance or presence. Learning agents have the ability to learn user's habits through observation, user feedback or training. Reasoning capability is very important for agents to operate in a decision-making capacity in complex, changing environment.

2.2 Mobile agents

Agents can be mobile by moving from machine to machine (form site to site). A mobile agent (MA) has a long-term memory (a suitcase) that includes all sites, which the MA visited, actions performed on each sites and the results of these actions. When it enters a new site a MA receives information about the site that might be useful for further decision-making (a briefing), such as local file system and databases. The suitcase and the briefing provide all the data needs of the agent [Bha96]. The agent server controls the migration of a MA to a new system by controlling a security and authentication of the MA, briefing the agent as it enters the system and executing the agent code [Bha96]. The MA uses hop instruction to move from one site to another.

Agent based systems are becoming one of the most important computer technologies, holding out many promises for solving real-world problems. An agent-based system may contain a single agent but the greatest potential lies in the application of multi-agent systems.

2.3 Multi-agent systems

While problems are often too complex to be solved by one intelligent agent the development is tending toward using multi-agent systems (MAS). MAS are agent groups where each component of intelligent behavior is delegated to a separate agent. Several agents that exist at the same time, share common resources and communicate with each other [Fer99]. MAS simplify problem solving by dividing the necessary knowledge into subunits to which an independent intelligent agent is associated and therefore every independent agent has the ability to solve a specific problem. The problem also has to bee discrete so that it can be divided into independent sub-problems. Individual agents are than assigned to solve a specific sub-problem. A partial global plan has to be created for tasks definitions.

2.4 Meta agents

Meta agents are a way to help agents observe the environment, evaluate alternatives and prescribe and schedule actions. In addition, strategies can be formulated and implemented not only within an agent but also among a group of agents. For any given problem, various strategies may be available. The main role of the meta agent is to sift through these strategies and make intelligent decisions. Eventually each agent will consult the meta agent prior to performing analyses; the agent will state the desired analyses and the utility of performing them. Subsequently, the meta agent will determine the feasibility of performing the actions by considering timing requirements and resource constraints. In addition, extra resources may be requested from the resource manager.

There are numerous applications where an agent needs to reason about the beliefs of another agent, as well as about the actions that other agents may take. Eiter [Eit99a] presents the concept of an agent program, and a language within which the operating principles of an agent can be declaratively encoded on top of imperative data structures is defined. In [Dix00] a certain belief data structures that an agent needs to maintain are introduced.

That extends the framework [Eit99b] so as to allow agents to perform meta reasoning.

In another work Stone discussed about the meta agent decisions on a strategy [Sto97]. It determines what behaviors of the agents would achieve this strategy and accordingly triggers those behaviors. By triggering only the behaviors appropriate to the current strategy, the meta agent also reduces behavior-behavior interactions. This is also in accordance with the layered architecture proposed in the paper. Higher level layers implement more abstract behaviors by selecting and activating the appropriate behaviors from the next level. The higher levels provide the strategic reasoning while the lower level provides reactivity to the system. This leads to the two most important questions the meta agent needs to answer:

- How to choose an appropriate strategy?
- How to characterize agent behaviors?

To allow coexistence of multiple agents Lakshmikumar proposed framework of the development of a reusable meta agent that manages these agents [Lak01]. The goals of this meta agent are:

- perform reasoning (to reason about agent autonomy): this is going to decide how much cooperation there needs to be between the agents;
- planning: plan actions;
- individual agent modeler: maintain individual agent state information;
- other agent modeler: maintain information on other agents and attempt to predict future behavior;
- conflict manager: classify conflicts and resolve them.

2.5 Use of agent systems in medicine

Agent systems and MAS are wide spread in all areas of user applications such as telecommunications, Internet, health care, tutoring systems, management systems, ecology, etc. They have also been useful in medicine [And00]. We will briefly highlight a few of these projects:

G. Lanzura, L. et al. [Lan99] at the University of Pavia, illustrated a methodology facilitating the development of interoperable intelligent software agents for medical applications and proposed a generic computational model for implementing them. That model may be specialized in order to support all the different information and knowledge related requirements of a Hospital Information System.

L.M. Camarinha-Matos and W. Vieira [Cam99] proposed an inexpensive support system for elderly people staying alone at home, allowing care and health centres to remotely observe and help them. It is based on the Internet and uses the multi-agent systems paradigm that includes both stationary and mobile agents.

M. Gnoth and I. Münich [Gno99] described the ChariTime project for the distributed scheduling of diagnostic and therapeutic appointments, based on multi-agent systems.

A. Boucher et.al. [Bou98] presented a multi-agent model for the analysis of living cells. The system is used particularly to study cell migration, for example the migration of tumor cells in response to treatment with antineoplastic drugs.

R. Freitas Jr. [Fre99] described medical nanorobots with tiny sensors and medical devices that will be capable of performing delicate, fine-grained operations within the human body.

An agent-based tutoring system for students of medicine was evolved at University of Southern California by Ganeshan et.al.[Gan00]

An agent-based approach to facilitate cooperative medical diagnosis was evolved at University College Galway in Ireland [Mul98]. It is achieved through monitoring patient record construction and by highlighting relevant diagnosis information.

3 Learning and intelligent agents

The machine learning community has paid increasing attention to problems of delayed reinforcement learning [Jaa94, Mca95]. These problems usually involve an agent that has to make a sequence of decisions, or actions, in an environment that provides feedback about those decisions. The basic loop followed in sequential decision making tasks such as these includes evaluating the current state, taking an action, and computing the new state. This loop is repeated until the system either reaches a goal state or recognizes that it will never terminate.

Research in multiple agent planning and control has been limited largely to the area of distributed artificial intelligence [Sto96] and artificial life [Dor96]. In distributed AI (DAI), several agents cooperate to achieve some goal or accomplish some task. The task is usually one of sufficient complexity that no single agent can accomplish the task alone. Because the agents cooperate, research in distributed AI has focused primarily on developing efficient procedures for communicating between the agents to enable the agents to develop the cooperative plans.

Although artificial life research does explore issues related to both cooperation and competition, its primary focus is on the emergence of intelligent behavior in a population of agents. For example, one area of application that has received considerable attention is the evolution of foraging behavior among artificial organisms (e.g., artificial ants) in the presence of predators. Also, migration patterns of artificial birds have been evolved. In none of these cases has behavior of individual agents been the focus of the research.

Recently, work has begun to appear that focuses on learning in MAS. Stone and Veloso provide a taxonomy of MAS by focusing on attributes such as agent homogeneity, communication, deliberative versus reactive control, and number of agents [Sto96]. Problems in MAS are distinct from problems in DAI and distributed computing, from which the field was derived, in that DAI and distributed computing focus on information processing and MAS focus on behavior development and behavior management. In addition, problems in MAS are distinct from problems in artificial life in that MAS still focus on individual behaviors and

artificial life focuses on population dynamics. So far, most work in learning and MAS has focused on multiple agents' learning complementary behaviors in a coordinated environment to accomplish some task, such as team game playing [Tam96], combinatorial optimization [Dor96], and obstacle avoidance [Gre91].

3.1 Learning agents with GP

Genetic programming (GP) and its variants have been applied to multi-agent learning. For instance, Koza used GP to evolve sets of seemingly simple rules that exhibit an emergent behavior. The goal was to genetically breed a common computer program, when simultaneously executed by all the individuals in a group of independent agent, i.e., the homogeneous breeding, that causes the emergence of beneficial and interesting higher-level collective behavior [Koz92].

Haynes proposed an approach to the construction of cooperation strategies based on GP for a group of agents [Hay95]. He experimented in the predator-prey domain, i.e., the pursuit game, and showed that the GP paradigm could be effectively used to generate apparently complex cooperation strategies without any deep domain knowledge.

Iba has applied GP-based multi-agent learning to the Tile World and proposed a co-evolutionary breeding scheme [Iba96]. Experimental results have shown the superiority of the co-evolutionary breeding over the two strategies, i.e., the homogeneous strategy and the heterogeneous strategy. In the co-evolutionary strategy, some individuals were expected to perform specialized tasks for different agents with generations.

4 Constructing decision graphs

Decision graph is an extension of a very well known decision tree representation [Qui93, Bre84, Pod02]. Similar to decision trees a decision graph contains attribute and decision nodes, where attribute nodes contain some kind of test of attributes' values and decision nodes serves to predict the solution (Figure 1, Figure 2). However, the decision graph principle is more flexible and more general than a decision tree. Since it contains also cycles, additional internal variables (different from attributes) can be added that help to process a temporal information, i.e., input can be represented in a time-series manner, which makes the decision graphs especially appropriate to deal with signals and continuous data. Decision trees are of course not able to process those kind of data.

A node in a decision graph contains a kind of transition rule that tells what edge to follow in a decision making process, based on the test of attributes' values and/or the state of internal variables. Transition rules can be very simple (as in decision trees) or more complex (each node contains a program). Since we decided to construct a decision graph with the help of evolving agents, the rules can not be too complex in order to maintain a simplicity of each of the participating agents. Therefore the rules are simple `if..then` statements, where the condition is a single attribute test or a single internal variable test. When composing two nodes (as a consequence of the JOIN agent) those simple statements are combined in a composed `if..then` statement. An example of a composed transition rule is presented on Figure 2.

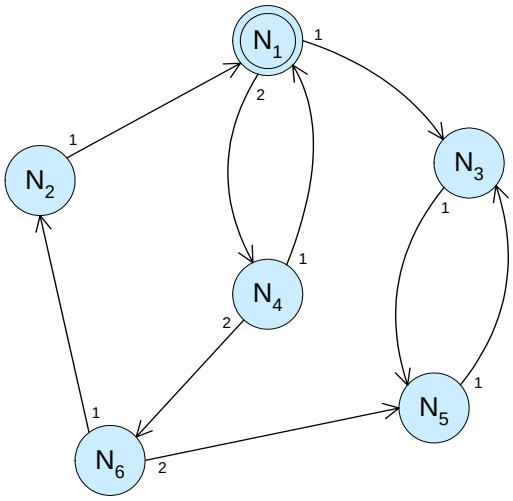

Figure 1. A simple decision graph. Every node contains a transition rule (Figure 2) that serves both as a test and/or as a decision class prediction. All edges from a single node are numbered, the numbers determine the next node based on the transition rule.

```
if (A3 > split) then
   if (INV2 < TRESHOLD2) then
      moveTo(3)
   else
      decision(CLASS1)
   endif
else
   moveTo(1)
endif
inc(INV2)
moveTo(1)
```

Figure 2. A composed transition rule. A_x is attribute `x`, `split` is a testing split, INV_x is internal variable, $THRESHOLD_x$ is a testing value for an internal variable, `moveTo(x)` indicates the transition to node `x`, $decision(CLASS_x)$ indicates the prediction of the decision class `x`, and $inc(INV_x)$ indicates the increase of the internal variable `x` by 1.

4.1 Two-leveled evolution process

The outline of the decision graph construction algorithm can be best described as a two-leveled evolution process. At the lower level decision graphs are being evolved. The evolution at this level starts by constructing an initial population of random decision graphs. For this purpose a random amount of decision nodes is created (a transition rule is initialized) which are then randomly connected with edges. In the continuation of the evolution at this

lower level in each generation participating agents modify the decision graphs. The quality of a decision graph is evaluated based on the accuracy of classification of training objects.

Naturally, because the construction of initial decision graphs is random, the classification accuracy in the early stages of evolution is low. Therefore, the quality of participating agents is essential in order to improve the predicting capabilities of the decision graphs. In this manner, the participating agents should improve to produce good results. To achieve the quality improvement in agents, they are also being evolved – and that is the second, the higher level of our global process. Each participating agent is evolved independently from the others by automatic programming approach with the *proGenesys* system. The quality of agents is not evaluated explicitly, but rather an universal complexity measure α is used that implicitly drives the agents to higher complexities.

In the Figure 3 a pseudo-code procedure for the two-leveled evolution process is presented.

```
initialize_agents()
repeat
   evolve_next_generations_of_agents()
   initialize_decision_graph()
   repeat
      apply_agents_to_modify_decision_graph()
      evaluate_decision_graph()
   until (num_generations > MAX_GENERATIONS)
   remember_the_best_decision_graph()
until (solution does not improve)
```

Figure 3. A pseudo code of the two-leveled evolution process of decision graphs construction. The inner `repeat..until` loop represents the lower level of the evolution process that changes decision graphs and the outer `repeat..until` loop represents the higher level of the evolution process that changes the agents.

4.2 Participating agents

Seven different agents are used in the process of decision graph evolution. We named those agents as: ADD, DELETE, MUTATE, JOIN, DISJOIN, PROTECT, and UNPROTECT. Each agent has its own function and works on the evolving decision graph independently from the other agents, according to its own procedure, defined by the outcome of genetic programming process in the current generation. In this way the decision graph is modified by those agents in order to become as accurate in classifying the training objects as possible. The functions of the participating agents are the following:

1. each ADD agent adds with certain probability: 1) a node (creates new transition rule for the new node), or 2) an edge (renumbering the existing connections);
2. each DELETE agent deletes with certain probability: 1) an edge (renumbering the remaining connections), or 2) a node (renumbering the connections of the connected nodes);
3. each MUTATE agent changes transition rule in a node with certain probability: 1) an attribute, 2) a split value, 3) an internal variable, 4) a threshold value for internal variables, 5) transition value, or 6) predicted decision;
4. each JOIN agent merges two selected nodes with certain probability, adjusting the transition rule and renumbering the new connections;
5. each DISJOIN agent separates a composed node into two connected nodes with certain probability, distributing the existing edges to either one new node or another;
6. each PROTECT agent protects with certain probability: 1) a node, and/or 2) an edge either against deletion, mutation, joining and/or disjoining;
7. each UNPROTECT agent unprotects with certain probability a protected: 1) node, and/or 2) edge;

5 Evolution of agents

All the agents are evolved with the use of automatic programming, a genetic programming technique for evolving programs in an arbitrary programming language, described with a context-free grammar. For this purpose we have used our evolutionary program generation tool called *proGenesys* [Pod99].

5.1 The kernel of *proGenesys*

The aim of the *proGenesys* tool is automatic generation of program code. We used genetic programming as the underlying principle of program generation. Generation of initial programs and basic evolutionary processes are quite similar, the most important difference represents the evaluation function that we used to determine the fitness of each individual. Since our intentions are to generate optimal programs performing some very complex task, we don't exactly evaluate evolved programs but rather use an universal complexity measure, namely the software complexity metrics α that is described later in the paper.

5.1.1 Generation of initial population

The first phase of genetic process is the generation of an initial population. Enough individuals have to be constructed to fulfill the whole population. Since later evolution depends quite a lot on initial population (especially its diversity), a great care was taken to implement a method for the construction of an individual.

For the construction of randomly generated programs, slightly modified Backus-Naur form (BNF) of programming language is used with some meta-symbols added, defining probabilities of transitions into specific branches of BNF structure, setting maximum recursion level of non-terminals extension, limiting the complexity of different program blocks, like expressions, etc. It is

important that the construction can start from within any BNF production, since it is also used later when mutation operator is applied. In this manner any BNF substructure can be generated when needed, like programming sentence, expression, etc.

An individual is internally represented as a syntax or derivation tree of a generated program (Figure 4). The syntax tree contains not only a program code but also a complete information on how this code was constructed from the starting symbol of the programming language's BNF. There are two types of nodes in a syntax tree. Internal nodes represent non-terminal symbols of BNF and show how each production was expanded to form the program. External or leaf nodes represent terminal symbols of BNF which actually construct the resulting program code. In this way program code can be extracted easily and syntax information is preserved throughout the evolution, which makes it easier to develop appropriate genetic operators.

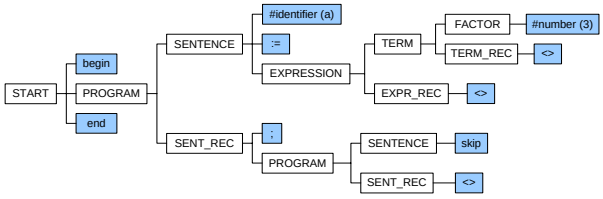

Figure 4. An example of a generated individual – a syntax tree.

5.1.2 Selection and fitness function

For the selection scheme we used a slightly modified exponential ranking selection method. After the evaluation of all individuals, they are sorted accordingly to their fitness score. Then we replace existing individuals from the worst to the best by creating new ones with crossover from two selected individuals, that still exist from the old population. When all the individuals are replaced, the new population is generated (there is still mutation to be applied).

For effective selection we have to define an adequate evaluation function, that determines the fitness score of each individual. This is the point where our approach differs the most from the other genetic programming applications. We don't try to exactly evaluate evolved programs, but rather use an universal complexity measure - our software complexity metric α. In this way individuals are evolved to very complex programs which are eventually evaluated through their performance upon decision graphs by measuring the effectiveness of the decision graphs in classifying the training objects.

5.1.3 Crossover

As the two individuals are selected from within the current population, a new solution is constructed by applying the genetic operator of crossover (Figure 5) and the constructed individual is placed in a growing new population. In order to perform a crossover operation, an appropriate crossover point has to be determined. For this purpose, a set of non-terminal symbols, contained in both selected individuals (parents), is computed and one of those symbols is chosen randomly. Then it is looked for such a node in both parent trees and offspring is created by concatenating a branch (a subtree) from the chosen node in second parent to the chosen node in first parent (see Figure 5). In this way the syntax correctness is preserved, since there is only a possible extension of BNF production replaced with another and the whole is still a correct program (considering that both parents were correct, what is actually the case, since the program generation algorithm guarantees only correct programs to be generated).

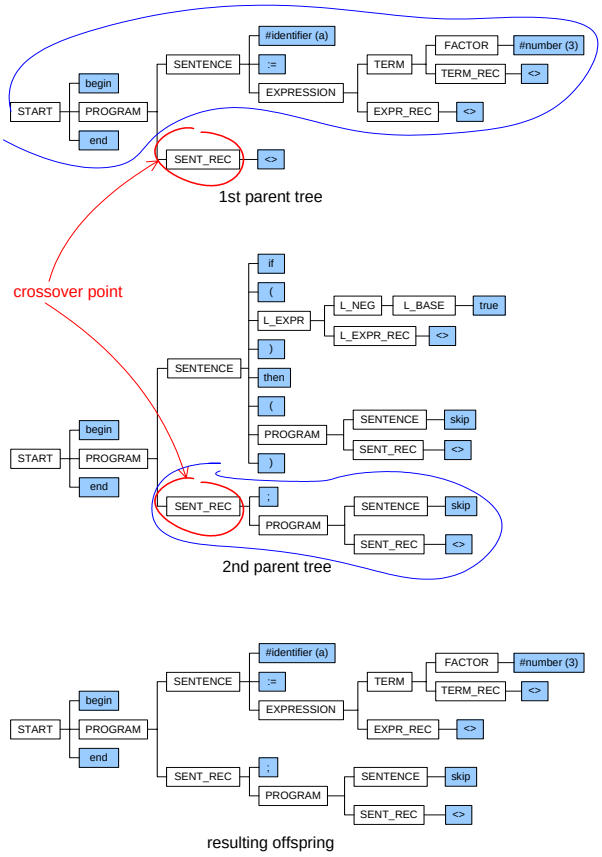

Figure 5. An example of a crossover. Circled parts of both syntax trees are combined into one offspring program.

5.1.4 Mutation

After a new individual is constructed by crossover, a genetic operator of mutation is applied (Figure 6) with certain probability. Mutation serves as a random change of an existing.

First the mutation point is randomly chosen in a given syntax tree. According to the selected node there are two possible situations. First, if an internal node was selected, representing a non-terminal symbol of BNF. In this case the existing extension of BNF production is replaced with a newly generated derivation. That is why

we mentioned the importance of a program generation algorithm to start with any BNF production, since here we send the chosen non-terminal symbol (from mutation point) and expect the algorithm to generate an adequate portion of a program code that is concatenated to the selected tree node instead of the existing part.

Second, if a special kind of external or leaf node was selected, representing a categorized terminal symbol of BNF. Such a categorized terminal is a number (category `#Number`) for example. In this case a new terminal is constructed so that it fits into specific category (for example, a new number is randomly chosen, replacing the existing one).

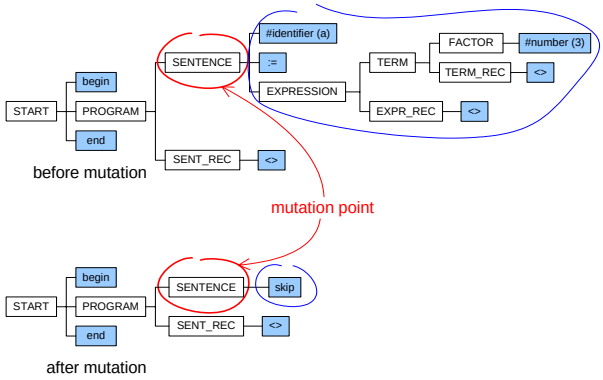

Figure 6. An example of a mutation. Circled part of first tree is mutated into the circled part of second tree.

5.2 Software complexity measure α

Many quantities have been proposed as measures of complexity. Gell-Man [Gel95] suggests there have to be many different measures to capture all our intuitive ideas about what is meant by complexity. Some of the quantities are computational complexity, information content, algorithmic information content, the length of a concise description of a set of the entity's regularities, logical depth, etc. (in contemplating various phenomena we frequently have to distinguish between effective complexity and logical depth - for example some very complex behavior patterns can be generated from very simple formula like Mandelbrot's fractal set, energy levels of atomic nuclei, the unified quantum theory, etc. - that means that they have little effective complexity and great logical depth). A more concrete measure of complexity, based on the generalization of the entropy, is correlation [Sch93], which can be relatively easy to calculate for a special kind of systems, namely the systems which can be represented as strings of symbols.

Computer programs are conventionally analyzed using the computational complexity or measured using complexity metrics. Another way to asses complexity is to use fractal metrics [Kok96, Kok99] or entropy based measure [Har89]. However, we can regard computer programs from the viewpoint of "complexity as a discipline" and according to that apply various possible complexity measures presented above. The fact that a computer program is a string of symbols, introduces an elegant method to asses the complexity – namely to calculate long range correlations between symbols, an approach which has been successfully used in the DNA decoding [Bul94] and on human writings [Sch93].

Our fractal measure α is based on char method, which is an extension of the method originally proposed by Schenkel [Sch93] for human writings. Like a human writing, a computer program can be seen as a string of symbols: letters, digits and some delimiting symbols – empty spaces are ignored. Using code table, where each of these symbols is represented by a binary sequence, the program is transformed into Brownian motion model (0's → step down, 1's → step up, see Figure 7), a base for the calculation of regression function *F(l)*:

$$F^2(l) \equiv \overline{[\Delta y(l)]^2} - \overline{\Delta y(l)}^2$$

$$\Delta y(l) = y(l_0 + l) - y(l)$$

where *Δy(l)* is relative difference between two points in Brownian motion model (Figure 7). The coefficient α is then calculated with the least squares method as the linear representation of the points on a double logarithmic scale [*ln (l), ln (F(l))*] and represents the complexity of a computer program (Figure 8).

According to above definition regression points [l_0, $F(l_0)$] are calculated from Brownian motion as follows:

$$F(l_0) = \sqrt{\frac{\sum_{l=1}^{S-l_0} \left(y(l+l_0) - y(l)\right)^2}{S-l} - \left(\frac{\sum_{l=1}^{S-l_0} y(l+l_0) - y(l)}{S-l}\right)^2}, \forall l_0 \in [1..\frac{S}{2}]$$

where *S* is the number of points in Brownian motion plot.

After the regression points are calculated, the coefficient α is then calculated with the least squares line as the linear representation of the points on a double logarithmic scale [*ln (l), ln (F(l))*]. As line crosses the axis at [*0, 0*] the line equation

$$y = a + bx$$

becomes simpler

$$y = bx$$

where *b* actually represents α.

From the method of least squares, *b* (or α in our case) is calculated as

$$b = \frac{n\sum_{i=1}^{n} x_i y_i - \left(\sum_{i=1}^{n} x_i\right)\left(\sum_{i=1}^{n} y\right)}{n\sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2}$$

Regarding the regression points [*ln (l), ln (F(l))*] the α is thus calculated as

$$\alpha = \frac{\frac{S}{2}\sum_{i=1}^{\frac{S}{2}} \ln(i)\cdot\ln(F(i)) - \left(\sum_{i=1}^{\frac{S}{2}} \ln(i)\right)\left(\sum_{i=1}^{\frac{S}{2}} F(i)\right)}{\frac{S}{2}\sum_{i=1}^{\frac{S}{2}} (\ln(i))^2 - \left(\sum_{i=1}^{\frac{S}{2}} \ln(i)\right)^2}$$

To better understand the process of calculating α the Figure 7 shows how the relative difference *Δy(l)* between two points in a Brownian motion model is calculated, and the Figure 8 shows the graphical representation of α on a double logarithmic scale of regression points [*ln (l), ln (F(l))*].

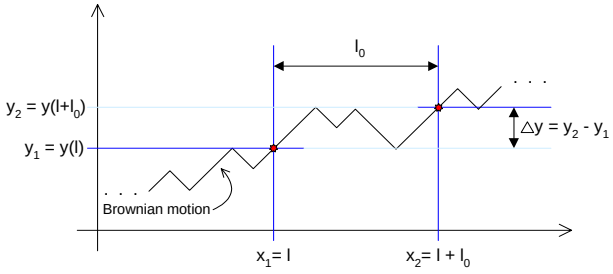

Figure 7. Calculation of regression curve points [*l, F(l)*] from a Brownian model plot.

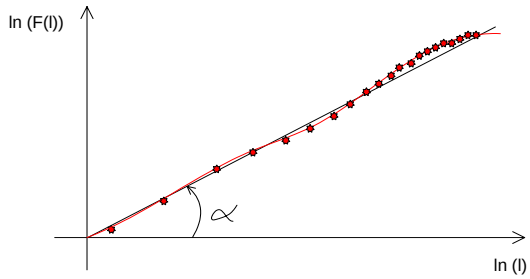

Figure 8. Representation of α on the regression curve points [*ln(l), ln(F(l))*] plot.

5.3 Programs for agents

In order to evolve the agents in accordance with our goal – to improve the classification capabilities of a decision graph – agents should be run by appropriate programs. The more obvious way to achieve this goal would be to define the programs for all the agents and let them work on the decision graphs. In this manner no second (higher) level of evolution would be necessary and the system would work faster. But of course, defining the agents is not a trivial task, and also we do not know what are the optimal programs for all the agents. Therefore, we decided to evolve the agents from the scratch with the use of the described *proGenesys* system.

For this purpose a language has to be defined first (a set of terminals, non-terminals and BNF productions) for each agent. We decided to keep the languages as simple as possible and therefore each language contains only few function calls to pre-defined functions (like `join()` for JOIN agent, `deleteNode()` or `deleteEdge()` for DELETE agent, etc.), simple condition statements (`if..then..else`, etc.) and simple expressions. A program for each agent is then interpreted to modify the decision graph in the lower level of evolution. An example of such a program for DELETE agent is presented in Figure 9.

```
if (random_condition == true) then
   node = selectNode()
   deleteNode(node)
else
   edge = selectEdge()
   deleteEdge(edge)
endif
```

Figure 9. An example of a simple program for the DELETE agent.

6 Application of the method and results

First we tested the performance of proposed classification method for training data and test data by the well-known iris data set [Fis36], which is not very complex. The iris data consists of 150 objects described by 4 continuous attributes and has three possible outcomes: Iris-setosa, Iris-versicolor, Iris-virginica. We selected 117 objects for training and the remaining 33 objects for testing. The average results over 5 runs are presented in Table 1.

Table 1. The results (accuracy) of iris data classification by evolved decision graphs (average over 5 runs).

	training data	test data
accuracy	96.58	93.94

An evolved decision graph for the classification of iris data is presented in Figure 10. It consists of 4 nodes, of which two contains classification of decision class. There is one cycle and two internal variables are used: one is a dummy and the other (`INV1`) actually plays an important role in classification process. It is interesting that there is only one decision statement for each class, what means that a classification for each class is made exactly once.

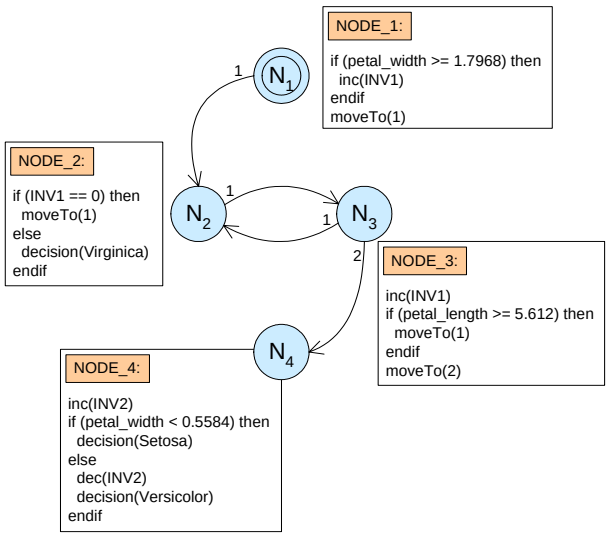

Figure 10. Evolved decision graph for the classification of iris dataset.

6.1 Mitral valve prolapse dataset

Because of the good results obtained for the iris data set, we decided to test a real-world medical problem of classifying mitral valve prolapse syndrome. Prolapse is defined as the displacement of a bodily part from its normal position. The term mitral valve prolapse (MVP) [And91, Dev89, Mar76], therefore, implies that the mitral leaflets are displaced relative to some structure, generally taken to be the mitral annulus. The silent prolapse is the prolapse which can not be heard with the auscultation diagnosis and is especially hard to diagnose. The implications of the MVP are the following: disturbed normal laminar blood flow, turbulence of the blood flow, injury of the chordae tendinae, the possibility of thrombus' composition, bacterial endocarditis and finally hemodynamic changes defined as mitral insufficiency and mitral regurgitation.

MVP is one of the most prevalent cardiac conditions, which may affect up to five to ten percent of normal population and one of the most controversial one. The commonest cause is probably myxomatous change in the connective tissue of the valvar liflets that makes them excessively pliable and allows them to prolapse into the left atrium during ventricular systole. The clinical manifestations of the Syndrome are multiple. The great majority of patients are asymptomatic. Other patients, however may present atypical chest-pain or supraventricular tachyarrhythmyas. Rarely, patients develop significant mitral regurgitation and, as with any valvar lesions, bacterial andocarditis is a risk.

Uncertainty persists about how it should be diagnosed and about its clinical importance. Historically, MVP was first recognized by auscultation of mid systolic "click" and late systolic murmur, and its presence is still usually suggested by auscultatory findings. However, the recognition of the variability of the auscultatory findings and of the high level of skill needed to perform such an examination has prompted a search for reliable laboratory methods of diagnosis. M-mod echocardiography and 2D echocardiography have played an important part in the diagnosis of mitral valve prolapse because of the comprehensive information they provide about the structure and function of the mitral valve.

Medical experts propose [And91, Mar76] that echocardiography enables properly trained experts armed with proper criteria to evaluate MVP almost 100%. Unfortunately however, there are some problems concerned with the use of echocardiography. The first problem is that current MVP evaluation criteria are not strict enough [Kok94]. The second problem is the incidence of the MVP in the general population and the unavailability of the expensive ECHO - machines to general practitioners. According to above problems we have decided to develop a decision support system enabling the general practitioner to evaluate the MVP using conventional methods and to identify potential patients from the general population.

6.2 Classification results and discussion

Using the Monte Carlo sampling method 900 children and adolescents representing the whole population under eighteen years of life have been selected. All of them were born in Maribor region and all were white. Routinely they were called for an echocardiography no matter of prior findings. From 900 selected 631 volunteers were successfully examined.

They all passed an examination of their health state in a form of a carefully prepared protocol specially made for the Syndrome of MVP. The protocol consisted of general data, mothers health, fathers health, pregnancy, delivery, post-natal period, injuries of chest or any other kind, chronic diseases, sports, physical examination, subjective difficulties like headaches, chest-pain, palpitation, perspiring, dizziness etc., auscultation, phonocardiography, ECG and finally ECHO. In that manner, 103 parameters were gathered that can possibly indicate the presence of MVP.

All 631 patient records were randomly divided into a training and a testing set. The average results over 10 runs, as obtained with the described evolutionary method, are presented in Table 2.

For the sake of comparison, we induced a traditional decision tree with C4.5 algorithm [Qui93]. It scored the following results for the test data set: accuracy 90.00%, sensitivity 63.64%, and specificity 91.60% (see Table 2).

Table 2. The results of MVP classification by evolved decision graphs (average over 10 runs) and C4.5.

	training data		test data	
	graph	C4.5	graph	C4.5
accuracy	92.21	94.21	86.92	90.00
sensitivity	94.83	54.24	72.73	63.64
specificity	91.87	96.3	88.24	91.60

Regarding the accuracy of classification our results are a bit worse than those obtained by classical decision tree induction method, both on the training and the test data. Also the specificity (percentage of correctly classified negatives, i.e. patients with no prolapse in our case) is better with C4.5 decision tree. On the other hand, the sensitivity (percentage of correctly classified positives, i.e. patients with prolapse our case) is better with our method. Because the number of positives is much smaller than negatives, it could be concluded that the decision graph produced by our method is more appropriate for classifying unbalanced data sets, which is very common in medicine.

Furthermore, decision trees induction methods like C4.5 are able to generate tree-like structures with their limited capabilities. Contrary our approach generates graphs, which have in medical environment a lot of advantages, like:

- classifying the cycle: diagnosis → treatment → outcome,
- revealing the relations between diagnosis, treatment and outcome,
- classification of temporal data like EEG, ECG, EMG, etc.

On the other hand, the use of our system (at least in current stage) is not as easy to use as C4.5 for example. There is some "overhead" needed to set up the method for a new classification task (like adaptation of programs induction, evolution of agents). Furthermore, the amount of computational resources is much higher than in a classical decision tree induction method, several runs are needed to evolve the proper agents and decision graphs. Finally, one further drawback of our method is the interpretability of the mined knowledge; because the nodes in our decision graphs are more complex, it is more difficult to interpret the decision graph model than the decision tree. However, the results are not a black box (as in the case of neural networks for example) and still allow an expert to validate them.

Regarding both the advantages and the drawbacks of our method, it can be concluded, that it is appropriate for difficult, unbalanced datasets, where even the smallest improvement in results is worth the higher effort in achieving this improvement. This is certainly the case in medicine, where human health and welfare is in question.

7 Conclusion

In the paper a new approach to the classification of medical data based on the meta agents system for the construction of decision graphs is presented. We applied it to the prediction of MVP, but being a general-purpose classification model it can be used for different kinds of classification tasks. Some reasons in favor of using a complexity-driven evolution of agents have been stated. The whole two-leveled evolution process of decision graphs construction is described and also the kernel of the *proGenesys* tool for automatic evolution of agent programs is described. Results of MVP classification by constructed decision graphs are compared with those obtained by traditionally constructed decision trees.

There are two essential contributions of the paper. The first one is the flexible decision graph approach to the classification, that is an extension of the well-known decision tree classifier. The second one is the complexity-driven evolution of agents, that can replace the explicit evaluation of individuals in the process of programs evolution. In this manner, the need for the definition of an appropriate fitness function is avoided.

An obvious drawback of our approach, when applied to a real-world problem, is somewhat lower classification accuracy (when compared to the decision trees), and especially the lower interpretability of the mined knowledge. Additionally, the proposed classification approach is more difficult to use than the majority of the known ones. On the other hand, there are important advantages, like good classification of unbalanced data sets, flexibility of the knowledge model, novelty of the complexity-driven approach to the evolution of agents, as stated in the application section of this paper.

In future we plan to reduce the effort needed to set up the described method for a new classification task. Another aspect that we want to explore is to transfer the implementation onto a grid system; greater computational power of a grid would increase the possibilities of evolution process – in this manner we hope to further improve the overall effectiveness and quality of the obtained results. If resources allow, we want to further develop the proposed concept.

References

[And91] Anderson HR et al, *Clinicians Illustrated Dictionary of Cardiology*, Science Press, London, 1991.

[And00] Andonyadis CG, *A Hybrid Architecture for Web-Based Personal Healthcare Support Agents*, PHD thesis, George Washington University, 2000.

[Bha96] Bharat K, Cardelli L, Migratory Applications, *Mobile Object Systems Toward the Programmable Internet: Second International Workshop*, pp. 131-148, Springer, 1997.

[Bou98] Boucher A, Doisy A, Ronot X, Garbay C, A society of goal-oriented agents for the analysis of living cells, *Artificial Intelligence in Medicine,* 4(1-2), pp. 183-199, 1998.

[Bre84] Breiman L, Friedman JH, Olsen RA, Stone CJ, *Classification and regression trees*, Wadsworth, USA, 1984.

[Bul94] Buldyrev SV et al., Fractals in Biology and Medicine: From DNA to the Heartbeat, *Fractals in Science* (Bundle A, Havlin S, eds.), Springer Verlag, 1994.

[Cam99]Camarinha-Matos LM, Vieira W, Intelligent mobile agents in elderly care**,** *Robotics and Autonomous Systems*, 27(1-2), pp. 59-75, 1999.

[Chu95] Chu-Caroll J, Carberry S, Communicating for Conflict Resolution in Multi-agent Collaborative Planning, in *Proc. of the International Conference on Multi-Agent Systems ICMAS'95*, 1995.

[Dec87] Decker KS, Distributed Problem Solving: A Survey, *IEEE Transactions on Systems, Man, and Cybernetics*, 17, pp. 729-740, 1987.

[Dev89] Devereoux R, Diagnosis and Prognosis of Mitral Valve Prolaps, *The New England Journal of Medicine*, 320(16), pp. 1077-1079, 1989.

[Dix00] Dix J, Subrahmanian VS, Pick G, Meta-agent programs**,** *The Journal of Logic Programming,* 46(1-2), pp. 1-60 , 2000.

[Dor96] Dorigo M, Maniezzo V, Colorni A, The Ant System: Optimization by a Colony of Cooperating Agents, *IEEE Transactions on Systems, Man and Cybernetics*, 26(1), pp. 1-13, 1996.

[Eit99a] Eiter T, Subrahmanian VS, Rogers TJ, Heterogeneous active agents, III: polynomially Implementable Agents, *Artificial Intelligence*, 117(1), pp. 107-167, 2000.

[Eit99b] Eiter T, Subrahmanian VS, Pick G, Heterogeneous active agents, I: semantics, *Artificial Intelligence*, 108(1-2), pp. 179–255, 1999.

[Fer99] Ferber J, *Multi-Agent System: An Introduction to Distributed Artificial Intelligence*, Addison Wesley Longman, 1999.

[Fis36] Fisher RA, The use of Multiple Measurements in Taxonomic Problems, *Annals Eugenics*, 7, pp. 179-188, 1936.

[Fre99] Freitas Jr. R, *Nanomedicine V1: Basic Capabilities*, Landes Bioscience Publishers, 1999.

[Gan00] Ganeshan R, Johnson WL, Shaw E, Wood BP, Intelligent tutoring systems, *Lecture notes in computer science*, Springer–Verlag, 2000.

[Gel95] Gell-Man M, What is complexity?, *Complexity*, 1(1), pp. 16-19, 1995.

[Gno99] Gnoth M, Münich I, *ChariTime Systementwurf*, Humboldt University Berlin, Dep. of Computer Science Working Paper, 1999.

[Gre91] Grefenstette J, Lamarkian Learning in Multi-agent Environments, *Proceedings of the Fourth International Conference of Genetic Algorithms*, pp. 303-310, Morgan Kaufmann, 1991.

[Har89] Harrison W, An Entropy-Based Measure of Software Complexity, *IEEE Transactions on Software*, 18(11), pp. 1025-1029, 1989.

[Hay95] Haynes T, Waiwright R, Sen S, Evolving a Team, *Working Notes of the AAAI-95 Symposium on Genetic Programming*, AAAI Press, 1995.

[Iba96] Iba H, Emergent Cooperation for Multiple Agents using Genetic Programming, *Parallel Problem Solving from Nature IV PPSN96*, 1996.

[Iba97] Iba H, Nozoe T, Ueda K, Evolving Communicating Agents based on Genetic Programming, *Proc. of the IEEE International Conference on Evolutionary Computation ICEC97*, 1997.

[Jaa94] Jaakkola T, Jordan M, Singh S, On the Convergence of Stochastic Iterative Dynamic Programming Algorithms, *Neural Computation*, 1994.

[Kok94] Kokol P, et al., Decision Trees and Automatic Learning and Their Use in Cardiology, *Journal of Medical Systems*, 19(4), 1994.

[Kok96] Kokol P, Brest J, Zumer V, Software Complexity - An Alternative View, *SIGPLAN*, 31(2), pp. 35-41, 1996.

[Kok99] Kokol P, Podgorelec V, Zorman M, Pighin M, Alpha - a generic software complexity metric, *Proc. of the ESCOM - SCOPE '99*, pp. 397-405, 1999.

[Koz92] Koza J, *Genetic Programming - On the Programming of Computers by means of Natural Selection*, MIT Press, 1992.

[Koz94] Koza J, *Genetic Programming II - Automatic Discovery of Reusable Programs*, MIT Press, Cambridge MA, 1994.

[Lak01] Lakshmikumar A, Meta-agents, http://zen.ece.ohiou.edu/~robocup/papers/HTML/SSST/node8.html, 2001.

[Lan99] Lanzola G, Gatti L, Falasconi S, Stefanelli M, A framework for building cooperative software agents in medical applications, *Artificial Intelligence in Medicine*, 16(3), pp. 223-249, 1999.

[Les95] Lesser VR, Multiagent Systems: An Emerging Subdiscipline of AI, *ACM Computing Surveys*, 27, pp. 340-342, 1995.

[Mar76] Markiewicz W, et al, Mitral valve Prolaps in One Hundred Presumably Young Females, *Circulation*, 53(3), pp. 464-473, 1976.

[Mca95] McCallum R, Instance-based Utile Distinction for Reinforcement Learning with Hidden State, *Proc. of the Twelfth International Conference on Machine Learning*, 1995.

[Mul98] Mulvihill C, Patel A, O'Meara T, Intelligent agents for collaborative diagnosis, *Proceedings of Medinfo98*, 9(1), pp. 232-236, 1998.

[Pod99] Podgorelec V, proGenesys - Program Generation Tool Based on Genetic Systems, *Proc. of the International Conference on Artificial Intelligence IC-AI'99*, pp. 299-302, CSREA Press, 1999.

[Pod02] Podgorelec V, Kokol P, Stiglic B, Rozman I, Decision trees: an overview and their use in medicine, *Journal of Medical Systems*, 26(5), pp. 445-463, 2002.

[Qui93] Quinlan JR, *C4.5: Programs for Machine Learning*, Morgan Kaufmann, San Mateo CA, 1993.

[Sch93] Schenkel A, Zhang J, Zhang Y, Long range correlations in human writings, *Fractals*, 1(1), pp. 47-55, 1993.

[Sto96] Stone P, Veloso M, Multiagent Systems: A Survey from a Machine Learning Perspective, *IEEE Transactions on Knowledge and Data Engineering*, 1996.

[Sto97] Stone P, Veloso M, Using decision tree confidence factors for multiagent control, *RoboCup-97: The First Robot World Cup*, 1997.

[Tam96] Tambe M, Teamwork in Real-world, Dynamic Environments, *1996 International Conference on Multiagent Systems*, AAAI Press, 1996.

[Woo95] Wooldridge M, Jennings NR, Intelligent Agents: Theory and Practice, *Knowledge Engineering Review*, Cambridge University Press, 10(2), 1995.