Anul XLVIII 2003

S T U D I A
UNIVERSITATIS BABEŞ-BOLYAI

INFORMATICA

2

Redacția: 3400 Cluj-Napoca, str. M. Kogălniceanu nr. 1 Telefon 405300

SUMAR – CONTENTS – SOMMAIRE

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLVIII**, Number 2, 2003

A NEW INTERFACE FOR REINFORCEMENT LEARNING SOFTWARE

GABRIELA ŞERBAN

Abstract. The field of Reinforcement Learning, a sub-field of machine learning, represents an important direction for research in Artificial Intelligence, the way for improving an agent's behavior, given a certain feed-back about its performance. In this paper we propose an original interface for programming reinforcement learning simulations in known environments. Using this interface, there are possible simulations both for reinforcement learning based on the states' utilities and learning based on actions' values (Q-learning).
Keywords: Reinforcement Learning, Agents.

1. Introduction

The interface is realized in JDK 1.4, and is meant to facilitate to develop software for reinforcement learning in known environments.

There are three basic objects:agents, environments and simulations.

The agent is the learning agent and the environment is the task that it interacts with. The simulation manages the interaction between the agent and the environment, collects data and manages the display, if any.

Generally, the inputs of the agent are perceptions about the environment (in our case states from the environment), the outputs are actions, and the environment offers rewards after interacting with it.

Figure 1 illustrates the interaction between the agent and the environment in a reinforcement learning task.

The reward is a number; the environment, the actions and perceptions are instances of classes derived from the *IEnvironment*, *IAction* and *IState* interfaces respectively. The implementation of actions and perception can be arbitrary as long as they are understood properly by the agent and the environment. It is obvious that the agent and the environment has to be chosen to be compatible with each other in this way.

Received by the editors: December, 10, 2002.

2000 *Mathematics Subject Classification.* 68T05.

1998 *CR Categories and Descriptors.* I.2.6 [**Computing Methodologies**]: Artificial Intelligence – *Learning*.

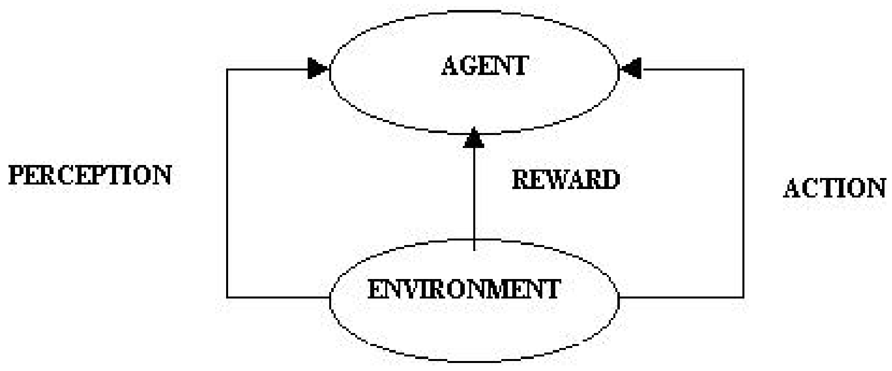


Figure 1. The interaction between the agent and the environment

The interaction between the agent and the environment is handled in discrete time. We assume we are working with simulations. In other words there are no real-time constraints enforced by the interface: the environment waits for the agent while the agent is selecting its action and the agent waits for the environment while the environment is computing its next state.

We assume that the agent's environment is a finite Markov Decision Process.

For using the interface, the user has to define the specialized object classes *HisState*, *HisEnvironment* and *HisAgent*, by creating instances for each. The agent and the environment are then passed to a simulation object (*CSimulation*), that initializes and interconnects them. Then, CSimulation::init() will initialize and execute the simulation.

If the agent learns the states' utilities, it has to be derived from the *AgentUtility* class, otherwise, if it learns the actions' values (Q-learning) it has to be derived from the *AgentQValues* class.

In the followings we present a prototypical example for a concrete agent.

(1) First, the user defines the class corresponding to a concrete state of the environment.
    public class *HisState* **implements** *IState*
    {...}
(2) Second, the user defines the class corresponding to the concrete environment in which the agent acts.
    public class *HisEnvironment* **implements** *IEnvironment*
    {...}
(3) The user defines the class corresponding to the concrete agent.
    public class *HisAgent* **implements** *AgentUtility*
    {

```
    public void actions(){...}
}
```

if the agent learns the states' utilities, respectively

```
public class HisAgent implements AgentQValues
{
    public void actions(){...}
}
```

if the agent learns the actions' values.
Using the method *actions()*, the agent perceives the actions that can be executed. In our approach, the agent's actions are numbered (starting from 1).

(4) Finally, the user defines the application class which initializes the simulation of learning process for the concrete agent in the concrete environment.

```
class Application {
    public static void main(String args[]){
            IEnvironment m=new HisEnvironment();
            RLAgent ag=new HisAgent();
//the agent perceives its actions
            ag.actions();
//instantiation for the object that realizes the simulation
            CSimulation s=new CSimulation(ag, m);
//on initialize the simulation - α, γ, ε, number of episodes
            s.init(0.01, 0.3, 0.1, 10);
//on display the policy
            s.policy(System.out);
}
```

We have to mention that the learning algorithms used for implementing the agents' behavior are the URU algorithm [4] for learning the states' utilities (values), respectively the SARSA algorithm [1] for Q-learning.

2. The Design of the Interface

The classes used for realizing the interface are the following:

- *IList* **INTERFACE**
  Defines the structure of a list of objects, having operations for managing the list: adding an element on a given position, removing an element from a given position, returning the number of elements from the list, returning an element from a given position.
- *IState* **INTERFACE**
  Defines the structure of a state from the environment (could have an explicit representation or an implicit one if the environment is unknown

and the agent has to retain a model of the environment). The methods of this class are for: returning a String with the member data of the class, testing the equality of two states.

- *IAction* **INTERFACE**

  Defines the structure of an action that the agent could execute. The methods of this class are for: returning a String with the member data of the class, testing the equality of two states.
- *Element* **ABSTRACT CLASS**

  Defines a generic element represented as a triplet (*IState*, *IAction*, *value*) needed for realizing the learning. Depending on the learning agent (learns the states' or the actions' values), *value* will represent the utility of the state *IState*, respectively the Q-value of the pair (*IState*, *IAction*).
- *Utility* **SUBCLASS of** *Element*

  Defines the class corresponding to an element (defined above) used in learning the states' utilities.
- *QValues* **SUBCLASS of** *Element*

  Defines the class corresponding to an element (defined above) used in learning the actions' values.

AGENT

The agent is the entity that interacts with the environment, receives perceptions (states) from it and selects actions. The agent learns by reinforcement and could have or not a model of the environment.

- *RLAgent* **ABSTRACT CLASS**

  Is the basic class for all the agents. The specific agents will be instances of subclasses derived from *RLAgent*. The methods of this class are:
  1. **void** *actions***()** **ABSTRACT METHOD**
     This function is given by the user for the specialized agent class and defines the list of actions that the agent could execute.
  2. **Element** *choose***(Element** *e***, Integer** *r***, double** *epsilon***, IEnvironment** *m***)** **ABSTRACT METHOD**
     This function is used in learning and allows the choice of the next element (having the type *QValues* or *Utility* , depending of the chosen learning type) to which the agent moves, starting from the current element *e*, in the environment *m* and choosing as a selection mechanism the *ϵ-Greedy* selection (*epsilon* is given as parameter). After this choice, the parameter *r* will contain the reward obtained by the agent.
     This method will have specific definition according to the learning method (Q-value, utility).

(3) **QValues** *next*(**IState** *s*, **IEnvironment** *m*) **ABSTRACT METHOD**
This function gives the agent's policy for moving after learning. If the object having the type *QValues* returned by the method contains the state *snext* and the action *a*, it means that the agent's policy is the following: from the state *s*, the agent will choose the action *a* and will move to the state *surm*.
This function has specific definition according to the agent's type, too.

(4) **Element** *initial*(**IState** *s*) **ABSTRACT METHOD**
The state *s* being the initial state of the environment, the method returns the initial element (*QValues* or *Utility*, corresponding to the learning's type) which will starts the learning. This method has specific definition according to the agent's type.

(5) **void** *learning*(**double** *alpha*, **double** *gamma*, **double** *epsilon*, **int** *episodes*, **IEnvironment** *m*)
Is the basic method which implements the learning algorithm of the agent in the environment *m*. There are given: the learning rate (*alpha*), the reward factor (*gamma*), the value for *epsilon* for the ϵ-Greedy selection mechanism, the number of training episodes (*episodes*).
This method is not abstract, is concretely defined in the class *RLAgent* (indifferent what is the learning's type, the learning method is the same).

- *AgentUtility* **ABSTRACT CLASS**
Is a subclass of the class *RLAgent*, being the entity which defines the behavior of an agent that learns by reinforcement based on the states' utilities. This class specializes the methods (2), (3) and (4) (defined in the superclass) according to learning based on states' utilities.
The method *actions()* is not defined in this class (that is why the class is abstract), but will be defined in the class corresponding to the specialized agent created by the user (and who can be an instance of a class derived from *AgentUtility*).
- *AgentQValues* **ABSTRACT CLASS**
Is a subclass of the class *RLAgent*, being the entity which defines the behavior of an agent that learns by reinforcement based on the actions' values. This class specializes the methods (2), (3) and (4) (defined in the superclass) according to the Q-learning method.
The method *actions()* is not defined in this class (that is why the class is abstract), but will be defined in the class corresponding to the specialized agent created by the user (and who can be an instance of a class derived from *AgentQValues*).

ENVIRONMENT

The environment basically defines the problem to solve. It determines the dynamic of the environment, the rewards and controls, the ending of the training process. In our approach, the environment will have an implicit representation as a space of sates (*IState*).

- *IEnvironment* **INTERFACE**

  Is the basic class for all environments. The specific environments will be instances of subclasses derived from *IEnvironment.* The environment classes defined by the user (subclasses of *IEnvironment*) will give specialized definitions for the following functions:

  (1) **boolean** *isValid*(**IState** *s*) **ABSTRACT METHOD**
  Is the function that verifies if a state *s* (represented explicitly or implicitly) is valid in its environment.

  (2) **IState** *initial*() **ABSTRACT METHOD**
  Is the method that returns the initial state of the environment (the state that will be used for initializing the learning).

  (3) **boolean** *isFinal*(**IState** *s*) **ABSTRACT METHOD**
  Is the method that returns the final state of the environment (the state that will be used by the agent for ending the training process). The final state could be given explicitly, or given implicitly by certain conditions.

  (4) **IState** *next*(**IState** *s*, **IAction** *a*, **Integer** *r*) **ABSTRACT METHOD**
  Is the main method of the interface *IEnvironment.* This method will be called by an instance of the class that simulates the learning (*CSimulation*), at each step of the simulation.
  This function determines the environment to make a transition from the current state *s* to a next state *surm*, after executing the specific action *a*. The state *surm* will be returned, the function supplying in the same time the reward *r* obtained after the transition.
  In the case that the action *a* could not be applied in the state *s*, the method returns *null.*

  (5) **double** *value*(**IState** *s*) **ABSTRACT METHOD**
  Is the method that gives the value of a state in the environment (the initial utility of the state and the initial Q-values). We considered that this value depends only on the current state, not on the selected action (in the case of Q-learning).
  This method will be used for initializing the learning process.

SIMULATION

- *CSimulation* **INTERFACE**

Is the basic object of the interface, that manages the interaction between the agent and the environment. Defines the *heart* of the interface, the uniform usage that all agents and environments are meant to conform to.

An instance of the simulation class is associated with an instance of an agent and an environment at the creation moment. This is made in the constructor of the class *CSimulation.* The methods of this class are for:

(1) **void** *init*(**double** *alpha*, **double** *gamma*, **double** *epsilon*, **int** *episodes*)
Is the method that initializes the simulation with the given parameters (is the function that starts the learning process of the agent).

(2) **void** *policy*(**PrintStream** *ps*)
Is the method that gives the moving policy for the agent, obtained at the end of the simulation (after the training process).
The class *CSimulation* keeps references to the instances of the agent and the environment. This facilitates cross-references of instances in case it is need.

```
public class CSimulation
{
    private RLAgent a; //reference to the agent's instance
    private IEnvironment m; //reference to the environment's instance
    ...
}
```

3. Experiment

In this section we illustrate the use of the interface on a concrete example. Let us consider the problem of a path-finding robot, whose goal is to learn (by reinforcement) to come out from a maze (moving from an initial to a final state).

We assume that:

- the maze has a rectangular form; in some positions there are obstacles; the agent starts in a given state and tries to reach a final (goal) state, avoiding the obstacles;
- from a certain position on the maze the agent could move in four directions: north, south, east, west (there are four possible actions);

For example, let us consider the environment from Figure 2. The state marked with 1 represents the initial state of the agent, the state marked with 2 represents the final state and the states filled with black contain obstacles (which the agent should avoid).

For using the interface, we defined the specialized classes for which we executed the simulation (*HisState*, *HisEnvironment* and *HisAgent*) (For lack of space

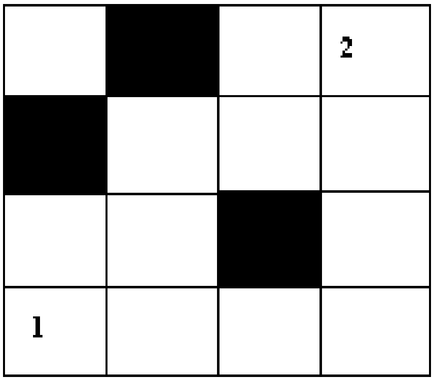


FIGURE 2. The agent's environment

the complete description of the classes may be found at the following URL: http://www.cs.ubbcluj.ro/∼gabis/agent.zip).

The moving policy learned by the agent after the training and reported by the *CSimulation* object is the same in both learning cases (states' utilities or Q-values). The learned path is: (4,1)-(3,1)-(2,2)-(2,3)-(1,3)-(1,4), respectively the sequence of actions that the agent should execute is: *North*, *East*, *North*, *East*, *North*, *East*.

4. Further Work

A further work for generalizing the interface would be the study of the case in which the agent's environment is a Hidden Markov Model [3].

References

[1] Sutton, R., Barto, A., G., Reinforcement learning, The MIT Press, Cambridge, England, 1998
[2] Serban, G., A Reinforcement Learning Intelligent Agent, Studia Universitatis "Babes-Bolyai", Informatica, XLVI (2), 2001, pp. 9–18
[3] Serban, G., Training Hidden Markov Models – a Method for Training Intelligent Agents, Proceedings of the Second International Workshop of Central and Eastern Europe on Multi-Agent Systems, Krakow, Poland, 2001, pp. 267–276
[4] Serban, G., A New Reinforcement Learning Algorithm, Studia Universitatis "Babes-Bolyai", Informatica, XLVIII (1), 2003, pp. 3–14

"Babeş-Bolyai" University, Cluj-Napoca, Romania
E-mail address: gabis@cs.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLVIII**, Number 2, 2003

STOCHASTIC OPTIMIZATION OF QUERYING DISTRIBUTED DATABASES II. SOLVING STOCHASTIC OPTIMIZATION

D. DUMITRESCU, C. GROŞAN, AND V. VARGA

Abstract. General stochastic query optimization (GSQO) problem for multiple join — join of p relations which are stored at p different sites — is presented. GSQO problem leads to a special kind of nonlinear programming problem (P). Problem (P) is solved by using a constructive method. A sequence converging to the solution of the optimization problem is built. Two algorithms for solving optimization problem (P) are proposed.

Keywords Distributed Databases, Query Optimization Problem, Genetic Algorithms, Evolutionary Optimization, Adaptive Representation.

1. Introduction

The aim of this paper is to solve the general stochastic optimization problem for the join of p relations, stored at p different sites of a distributed database. In Part I the general stochastic optimization problem, was reduced to the following constrained nonlinear programming problem (P):

Let(X, d) be a compact metric space and

$$f_1, ..., f_p : X \to R_+$$

continuous, strictly positive functions.

The optimization problem (P) is thus:

$$(P) \begin{cases} \text{minimize } y, y \in R \\ \quad \text{subject to:} \\ y > 0, \\ f_1(x) \le y, \\ \vdots \\ f_p(x) \le y. \end{cases}$$

Received by the editors: July 3, 2003.

2000 *Mathematics Subject Classification.* 68P15, 68T99.

1998 *CR Categories and Descriptors.* C.2.4 [**Computer Systems Organization**]: Computer-Communication Networks – *Distributed Systems*; H.2.4 [**Information Systems**]: Database Management – *Systems*.

In this Part a constructive method to solve this problem is proposed. A theorem which demonstrates that the nonlinear optimization problem (P) has at least one solution is proved in Section 2.

The *Constructive Algorithm* (CA) given in Section 3 implements the method of Section 2. The *Refining Algorithm* (RA) can optimize the solution given by the Constructive Algorithm. RA starts with a minimum point x_{min} and searches for a better solution in the $[x_{\min} - \varepsilon, x_{\min}]$ interval and then in $[x_{\min}, x_{\min} + \varepsilon]$, where ε is a problem parameter.

2. A constructive method for solving general stochastic query problem

Now we are ready to give a constructive method for solving problem (P). This method generates a sequence converging to a solution of the problem (P). Theorem 2.1 ensures that the constructed sequence really converges towards a solution of the optimization problem (P).

Let $f : X \to R$ be the function defined by

$$f(x) = \max\{f_1(x), ..., f_p(x)\}.$$

and y_0 the global minimum value of the function f, i.e.

$$y_0 = \min_{x \in X} f(x).$$

Let $A_1 \subset A_2 \subset A_3 \subset \ ... \ \subset A_n \subset \ ...$ be a sequence of finite subsets of X such that $\overset{\infty}{\underset{n=1}{\cup}} A_n$ is dense (see for instance Rudin, 1976) in X, i.e. $\overline{\cup A_n} = X$ equivalent to the fact, that for $\forall x \in X$, $\exists x_n \in \underset{n \in N}{\cup} A_n$ such that $x_n \to x$.

We consider

$$\begin{array}{rcll} A_1 & = & \{u_1, u_2, ..., u_{q_1}\}, & u_i \in X, i = 1, \ldots q_1, \\ A_2 & = & \{v_1, v_2, ..., v_{q_2}\}, & v_j \in X, j = 1, \ldots q_2, \\ \vdots & & & \\ A_n & = & \{w_1, w_2, ..., w_{q_n}\}, & w_k \in X, k = 1, \ldots q_n, \end{array}$$

where $q_i \in N^*, i = 1, \ldots, n$ and $q_n \to \infty$.

Let us consider the sequence $(y_n)_{n \geq 1}$ defined as folows:

$$\begin{array}{rcl} y_1 & = & \min\{\max\{f_1(u_1), f_2(u_1), ..., f_p(u_1)\}, ..., \max\{f_1(u_{q_1}), f_2(u_{q_1}), ..., f_p(u_{q_1})\}, \\ y_2 & = & \min\{\max\{f_1(v_1), f_2(v_1), ..., f_p(v_1)\}, ..., \max\{f_1(v_{q_2}), f_2(v_{q_2}), ..., f_p(v_{q_2})\}, \\ \vdots & & \\ y_n & = & \min\{\max\{f_1(w_1), f_2(w_1), ..., f_p(w_1)\}, ..., \max\{f_1(w_{q_n}), f_2(w_{q_n}), ..., f_p(w_{q_n})\}. \end{array}$$

It is easy to see that sequence $(y_n)_{n \geq 1}$ is monotone decreasing and bounded. Therefore the sequence is convergent.

With respect to the convergent sequence $(y_n)_{n\geq 1}$ we can state the following Theorem.

Theorem 2.1 The sequence $(y_n)_{n\geq 1}$ converges to a solution of the problem (P).

Proof. We have

$$y_n \geq f(x_0)$$

because x_0 is the global minimum of the function f. Therefore, if $y_n \to y^*$ we have

$$y^* \geq f(x_0).$$

We distinguish two cases. First case corresponds to

$$y^* = f(x_0).$$

In this case is nothing to demonstrate. The second case corresponds to the situation

$$y^* > f(x_0).$$

We prove that this case it is impossible.

Because the set $\overset{\infty}{\underset{n=1}{\cup}} A_n$ is dense in X and the function f is continuous it results that there exists a sequence $(x_n) \subset \overset{\infty}{\underset{n=1}{\cup}} A_n$ such that

$$x_n \to x_0 \quad \text{and} \quad f(x_n) \to f(x_0).$$

Without loss of generality we may suppose that

$$x_1 \in A_1, ..., x_n \in A_n, ...$$

But we have:

$$y_n = \min\{\max\{f_1(w_1), \ldots, f_p(w_1)\}, \ldots, \max\{f_1(w_{q_n}), \ldots, f_p(w_{q_n})\}\}$$

and

$$f(x_n) = \max\{f_1(x_n), \ldots, f_p(x_n)\}$$

Therefore we have:

$$f(x_n) \geq y_n,$$

for every $n \in N^*$.

If $n \to \infty$ we have $f(x_n) \to f(x_0)$ and $y_n \to y^*$, so we obtain

$$f(x_0) \geq y^*,$$

which is a contradiction with the assumption $y^* > f(x_0)$. Therefore we obtained $y^* = f(x_0)$. This completes the proof. $\square$

Remark. From the construction above we can see that for every $n \in N^*$, there exists an index $i_n \in \{1, ..., q_n\}$such that

$$y_n = \max\{f_1(w_{i_n}), ..., f_p(w_{i_n})\}.$$

In this way we obtain a sequence $(w_{i_n})_{n\geq 1}$. It is obvious that each accumulation point of the sequence $(w_{i_n})_{n\geq 1}$ is a solution of the problem (P).

3. Solving problem (P_p) using the proposed constructive method

In the case of solving problem (P_p) using Theorem 2.1 we have

$$X = [0,1]^n .$$

In order to obtain an approximate solution of problem (P_p) in the Constructive Algorithm we take a uniform grid G of the hypercube [0,1]^k.

We may choose the sets $(A_i)_{i \in N^*}$ in the folowing way:

$$\begin{aligned}
A_1 &= \left\{ \left(\frac{i_0}{n}, \frac{i_1}{n}, \dots, \frac{i_n}{n} \right) | i_0, i_1, \dots, i_n \in \{0,1,\dots,n\}, i_0 < i_1 < \dots < i_n \right\}, \\
A_2 &= \left\{ \left(\frac{j_0}{2n}, \frac{j_1}{2n}, \dots, \frac{j_{2n}}{2n} \right) | j_0, j_1, \dots, j_{2n} \in \{0,1,\dots,2n\}, j_0 < j_1 < \dots < j_{2n} \right\}, \\
&\vdots \\
A_k &= \left\{ \left(\frac{l_0}{2^{k-1}n}, \frac{l_1}{2^{k-1}n}, \dots, \frac{l_{2^{k-1}n}}{2^{k-1}n} \right) | l_0, l_1, \dots, l_{2^{k-1}n} \in \left\{0,1,\dots,2^{k-1}n\right\}, \right. \\
&\quad \left. l_0 < l_1 < \ldots < l_{2^{k-1}n} \right\}.
\end{aligned}$$

Our grid is that induced by $A_1, A_2, \dots, A_k$. The sets $(A_i)_{i \in N^*}$ constructed in the above way verify the conditions of Theorem 5.1 of Part I of this paper. For our purposes we may consider $n = 10$.

For each point of the grid G we compute the values $f_s, s = 1, \dots, p$. Choosing the maximum$f_s, s = 1, \dots, p$, we ensure that each inequality in the problem (P_p) holds. Problem solution will be the minimum of all selected maximums.

The previous considerations enable us to formulate an algorithm for solving problem (P_p). This technique will be called *Constructive Algorithm* (CA) and may be outlined as below.

Constructive Algorithm

```
Input:
    $n$                                // the number of divisions;
    Functions $f_1$,$f_2$...,$f_p$       // express the problem constraints.
begin
Initializations:
    $h = \frac{1}{n}$                  // the length of one division;
    $valx_j = 0, j = 1, ..., k$        // initial values for $x_j$;
    for $s = 1$ to $p$ do              // initial values for functions $f_s$
        $valf_s = f_s(valx_1,\ valx_2, \dots, valx_k)$
    end for
    $valmax = \max\{valf_s, s = 1, ..., p\}$
    $valmin = valmax$
    for $j = 1$ to $k$ do                 // in $xmin_j$ we store the $x_j$ values for which we
      $xmin_j = valx_j$                   // have the minimum of $f_s$
    end for
Constructing the grid:
    for $i_1 = 1$ to $n$ do
      $valx_1 = i_1 * h$
      for $i_2 = 1$ to $n$ do
        $valx_2 = i_2 * h$
          ⋮
          for $i_k = 1$ to $n$ do
            $valx_k = i_{k*}h$
            for $s = 1$ to $p$ do         // calculate the values for functions $f_s$ for
              $valf_s = f_s(valx_1, valx_2, \ldots, valx_k)$     // the current values of $x_j$
            end for
            $valmax = \max\{valf_s, s = 1, ..., p\}$
            if $(valmax < valmin)$ then
              $valmin = valmax$
              for $j = 1$ to $k$ do       // store in $xmin_j$ the new $x_j$ values
                $xmin_j = valx_j$         // for which we have the
              end for                     // minimum of $f_s$
            end if
          end for // $i_k$
            ⋮
        end for // $i_2$
      end for // $i_1$
end
```

Remark. *valmin* denote the minimum value of Δ_1 from problem (P_p) and $xmin_j$, $j = 1, ..., k$ denote the values for x_j, $j = 1, ..., k$ for which the minimum is reached.

The Constructive Algorithm should be repeated for a new value of n, so that the divisions have to include the old divisions, in this way we obtain a new subset A_i of the set X.

Solution obtained by the Constructive Algorithm can be refined using the *Refining Algorithm* (RA).

Let us denote by $(x_{\min 1}, x_{\min 2}, ..., x_{\min k})$ the minimum point obtained by the Constructive Algorithm. Let us define the vectors $x_{\min} - \varepsilon$, $x_{\min} + \varepsilon$:

$$\begin{aligned} x_{\min} - \varepsilon &= (x_{\min 1} - \varepsilon, x_{\min 2} - \varepsilon, ..., x_{\min k} - \varepsilon), \\ x_{\min} + \varepsilon &= (x_{\min 1} + \varepsilon, x_{\min 2} + \varepsilon, ..., x_{\min k} + \varepsilon) \quad . \end{aligned}$$

Initially Refining Algorithm searches for a better minim in the interval: $[x_{\min} - \varepsilon, x_{\min}]$. Then it searches in $[x_{\min}, x_{\min} + \varepsilon]$, where ε is a problem parameter. In case of found a better minim (to the left, or to the right) the algorithm will continue to search refining the grid by division by 2. Let *IterNr* be the maximum allowed number of iterations.

Refining Algorithm can be outlined as follows.

Refining Algorithm

```
Input:
    $n$                        // the number of divisions;
    $eps$                      // the accepted error;
    $IterNr$                   // the number of iterations;
    $xmin_j, j = 1, ..., k$    // a minimum point obtained with algorithm CA;
Initializations:
    $h = \frac{1}{n}$                  // the length of one division;
    for $s = 1$ to $p$ do      // values for functions $f_s$;
      $valf_s = f_s(xmin_1, xmin_2, \ldots, xminx_k)$
    end for
    $valmin = \max\{valf_s, s = 1, ..., p\}$
    for $j = 1$ to $k$ do      // in $xminr_j$ we store the $x_j$ values for which we
      $xminr_j = xmin_j$       // have the minimum of $f_s$
    end for
Refining the minimum:
    while $h >= eps$ do
      for $iter = 1$ to $IterNr$ do
        for $j = 1$ to $k$ do
          while found a better minimum to the left do
            if $xmin_j - h > 0$ then
              $xmin_j = xmin_j - h$
              $valmax = \max\{f_s(xmin_1, ..., xmin_k), s = 1, ..., p\}$
              if $(valmax < valmin)$ then
                $valmin = valmax$            // a new minimum was found;
                for $j = 1$ to $k$ do        // store in $xminr_j$ the new $x_j$
                  $xminr_j = xmin_j$         // values for which we have the
                end for // minimum of $f_s$;
                reloop while
              end if
            end if
          end while // found to the left
          while found a better minimum to the right do
            if $xmin_j + h > 0$ then
              $xmin_j = xmin_j + h$
              $valmax = \max\{f_s(xmin_1, ..., xmin_k), s = 1, ..., p\}$
              if $(valmax < valmin)$ then
```

```
                    $valmin = valmax$            // a new minimum was found;
                    **for** $j = 1$ **to** $k$ **do**         // store in $xminr_j$ the new $x_j$
                      $xminr_j = xmin_j$             // values for which we have the
                    **end for** // minimum of $f_s$;
                    **reloop while**
                  **end if**
                **end if**
              **end while** // found to the right
            **end for** // $j$
          **end for** // $iter$
          $h = h/2$ // refine the division;
        **end while** // $h >= eps$
```

Algorithms CA and RA can be used to solve the general stochastic optimization problem (P). The problem of four relations join is formulated as the problem (P_1) of Part I, which is a particularization of general problem (P).

Numerical experiments for solving problem (P_1) using the Constructive Algorithm and Refining Algorithm are presented in Part III.

References

[1] C. J. Date (2000): *An Introduction to Database Systems*, Addison-Wesley Publishing Company, Reading, Massachusetts.

[2] R. F. Drenick (1986): *A Mathematical Organization Theory*, Elsevier, New York.

[3] P. E. Drenick, R. F. Drenick (1987): *A design theory for multi-processing computing systems*, Large Scale Syst. Vol. 12, pp. 155-172.

[4] P. E. Drenick, E. J. Smith (1993): *Stochastic query optimization in distributed databases*, ACM Transactions on Database Systems, Vol. 18, No. 2, pp. 262-288.

[5] D. Dumitrescu, C. Grosan, M. Oltean (2001): *A new evolutionary adaptive representation paradigm*, Studia Universitas “Babes-Bolyai”, Seria Informatica, Volume XLVI, No. 1, pp. 15-30.

[6] C. Grosan, D. Dumitrescu (2002): *A new evolutionary paradigm for single and multiobjective optimization*, Seminar on Computer Science, “Babes-Bolyai” University of Cluj-Napoca.

[7] J. Kingdon,L. Dekker (1995): *The shape of space*, Technical Report, RN-23-95, Intelligent System Laboratories, Department of Computer Science, University College, London.

[8] S. LaFortune, E. Wong (1986): *A state transition model for distributed query processing*, ACM Transactions on Database Systems, Vol. 11, No. 3, pp. 294-322.

[9] T. Markus, C. Morosanu, V. Varga (2001): *Stochastic query optimization in distributed databases using semijoins*, Annales Universitatis Scientiarum Budapestinensis de Rolando Eötvös Nominatae Sectio Computatorica 20, pp. 107-131.

[10] M. T.Özsu, P. Valduriez (1999) : *Principles of Distributed Database Systems*, Prentice-Hall.

[11] R. Ramakrishnan (1998): *Database Management Systems* WCB McGraw-Hill.

[12] W. Rudin(1976): *Principles of Mathematical Analysis*, McGraw-Hill, New York.

[13] J. D. Ullman (1988): *Principles of Database and Knowledge-Base Systems,* Vol. I-II, Computer Science Press.

[14] V. Varga (1998): *Stochastic optimization for the join of three relations in distributed databases I. The theory and one application*, Studia Universitas "Babes-Bolyai", Seria Informatica, Volume XLIII, No. 2, pp. 37-46.

[15] V. Varga (1999): *Stochastic optimization for the join of three relations in distributed databases II. Generalization and more applications*, Studia Universitas "Babes-Bolyai", Seria Informatica, Volume XLIV, No. 1, pp. 55-62.

E-mail address: `ddumitr@cs.ubbcluj.ro`

E-mail address: `cgrosan@cs.ubbcluj.ro`

E-mail address: `ivarga@cs.ubbcluj.ro`

Department of Computer Science, Faculty of Mathematics and Computer Science, Babeş-Bolyai University, Cluj-Napoca, Romania

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLVIII**, Number 2, 2003

EVOLUTIONARY CLUSTERING USING AN INCREMENTAL TECHNIQUE

R. GORUNESCU AND D. DUMITRESCU

Abstract. Since the various clustering methods developed over the time have failed to prove their flawless efficiency in the field, it might be that evolutionary computation holds the solution to this issue as well.

The goal of this paper is to present such an evolutionary technique with a classical clustering engine behind it.

Keywords: incremental clustering, evolutionary computation, genetic algorithms, merging, splitting, weighted similarity measures

1. Introduction

A new evolutionary clustering technique is proposed. This method represents an evolutionary variant of the incremental clustering technique.

Incremental clustering (IC)[6] is a powerful clustering method that is of great interest mainly because the number of clusters is not specified. This feature is very important in the field of unsupervised learning. Therefore, instances are added one by one forming a tree, starting with an empty root node. The best location for the new instance or the best restructuring of the part of the tree affected by it is determined by several operators, operators whose diversity leads us to the second reason behind the success of the IC method. The ordinary Euclidean distance - usually used for building an objective function - is replaced by a function called *category utility* which measures the quality of the partitioning.

2. Incremental Clustering

The idea behind the IC algorithm is quite simple. We start with an empty root. Instances are added sequentially until there are none remaining, as follows: for each instance we compute the category utility of placing it into an existing leaf versus the category utility of forming a leaf by itself. Whichever is better will determine the location of that instance. This is the general approach, but if we

Received by the editors: September 12, 2003.

2000 *Mathematics Subject Classification.* 62H30, 68T20.

1998 *CR Categories and Descriptors.* I.5.3 [Pattern Recognition]: Clustering – *Clustering algorithms*; I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods and Search – *Heuristic methods*.

were to continue in this manner the resulting clustering would be dependent on the order in which instances were considered. Therefore, we move on to the next two operators that intervene now in the process. Firstly, there is the merging operator - that is combining two classes into a single one before the new instance is added to the resulting leaf - and secondly, the reverse one, that is the splitting operator - dividing a class into two. These two operators have proven to be extremely important in balancing the possible negative effects of the above mentioned ordering of instances.

2.1. ***Category utility*** **criterion.** A function to measure the quality of the clustering with a complex role is considered. The proposed function maximizes both the probability that instances in the same class have common attribute values and the probability that instances from different classes do not.

Let us consider the following notations:

$P(A = v|C)$ is the probability that an instance has value v for its attribute A, given that it belongs to class C. The higher the probability, the more likely instances in the same class will have attribute values in common.

$P(C|A = v)$ is the probability that an instance belongs to class C, given that it has value v for its attribute A. The higher the probability, the less likely instances from different classes will have common attribute values.

$P(A = v)$ is a weight of the fact that frequently occurring attribute values have a stronger influence on the evaluation.

Let $C_1, ..., C_k$ be the current partition. The category utility function $U(C_1, ..., C_k)$ is the quantity defined as follows:

$$U(C_1, ..., C_k) = \sum_C \sum_A \sum_v P(A = v)P(A = v|C)P(C|A = v),$$

where the first sum is taken with respect to all clusters, the second one with respect to all attributes of the members in class C and the third with respect to the attribute values.

However, this expression is not the one that is being used in practice. Instead, we will use a slightly changed form. This new expression is obtained by applying the Bayes formula for conditional probabilities, that is

$$P(A \cap B) = P(A|B)P(B) = P(B|A)P(A),$$

and thus by simplying the first expression using the above formula, we obtain:

$$\sum_C \sum_A \sum_v P(A = v|C)^2 P(C)$$

The final form of the category utility will measure the amount by which information about what cluster the current instance is in does make a difference, compared to the situation of not knowing anything about the cluster structure, summed over all the clusters by their probabilities. Finally it is divided by the number of clusters to discourage the phenomenon that each instance would be put in its own cluster.

3. A new evolutionary clustering algorithm

The drawbacks of the IC method are not particularly disturbing at large, with only one exception. To what extent is the final result dependent on the order of examples? Are the two operators - splitting and merging - sufficient to prevent this dependence?

The present method is desired to take care of this aspect through the techniques of evolutionary computation, on the one hand, and to take advantage of the characteristics of a powerful method, the incremental clustering, on the other hand.

The incremental nature of the IC method will be preserved precisely. The mechanisms of evolutionary algorithms will provide the several parallel possibilities of ordering the instances. The effect of the merging and splitting operations will be identical to that of the original method by means of the recombination and mutation operators. Finally, the expression of the fitness function will be inspired from the category utility criterion, but dwelt upon more from a similarity comparison between instances point of view rather than from a probabilistic one.

3.1. **Representation. Initial population.** The value of a gene will represent the cluster number of the instance labelled with the position of that gene. That is, if we denote by c the current chromosome, then c_i will give the number of the cluster in which the i-th instance will be, $i = 1, ..., m$, where m is the number of instances of the specified database. For example, if we have four objects, then the chromosome (1,3,3,2) means that instance one is in a cluster, instance four in another cluster, and instances two and three in the third cluster.

The initial population will be made up of chromosomes with a single 1-valued arbitrary position, that is for every chromosome we take randomly an instance and put it in the first cluster. In this way, the algorithm starts to offer several parallel possibilities of ordering the data, and continues furthermore in this sense by the means of a special variation operator.

3.2. **Fitness function.** First of all, we have to define the similarity measure between two instances of our considered database, since our function is built upon its expression. We have used a weighted similarity measure, since each attribute of our data has a different degree of importance in the field they are extracted from.

$$distance(a, b) = \sum_{k=1}^{n} compare(a_k, b_k),$$

where a and b are the two instances and n represents the number of attributes. At this point there are two cases:

(i) If we are dealing with numerical attributes, the difference between the two attributes is the square weighted Euclidian similarity measure, that is:

$$compare(a_k, b_k) = (a_k - b_k)^2 weight_k,$$

where $weight_k$ is a positive number specifying the importance of attribute k.

(ii) In the other case, of the nominal attributes, we have considered their representation as fuzzy. Therefore, for the difference between such attributes, the max-min distance specific to fuzzy data is considered:

$$compare(a_k, b_k) = \max(\min_{i=1}^{n_k})(a_k^i, 1 - b_k^i) weight_k,$$

where n_k is the number of values for the k-th attribute of the chromosome.

For the expression of the fitness value, we will act as it follows.

Let c be the current chromosome and we would like to compute its performance. Then

$$eval(c) = \frac{\sum\limits_{Clst=1}^{k} \sum\limits_{i,j=1,\ldots,m, i<j, c_i=c_j=Clst} distance(instance_i, instance_j)}{k},$$

where $instance_i$ represents the i - th instance in the database and k represents the number of clusters denoted by that chromosome.

If a gene with a unique value in that chromosome is found, its penalty (instead of the *distance* function in the above formula) for forming a cluster of its own will be 1.

The division by the number of clusters prevents the phenomenon of too crowded clusters.

As it can be easily seen, this performance function is somewhat similar to the expression of the category utility in IC, with the significant difference that, while category utility has to be maximized, we are trying to minimize our fitness function.

Another fitness evaluation can be considered independent of the category utility criterion, but still considering the basic idea of clustering, that is minimizing intra-class distance and maximizing inter-class distance. Therefore, we are led to the following multi objective optimization problem (MOEA):

$$f_1(c) = \frac{\sum\limits_{k} \sum\limits_{c_i=c_j=c_k} d(instance_i, instance_j)}{|c_k|^2} \to min$$

and

$$f_2(c) = \frac{\sum\limits_{k} \sum\limits_{c_i=c_k, c_j \neq c_k} d(instance_i, instance_j)}{|c_k|^2} \to max$$

Standard MOEAs [1] can be used for solving our problem.

The output of a MOEA is a set of feasible solutions, the optimal Pareto solutions [1].

To avoid difficulty in choosing a single Pareto solution, we propose to combine the objective functions f_1 and f_2 in a unique criterion function:

$$F(c) = k_1 f_1 + k_2 \frac{1}{f_2}$$

We are led to $F \rightarrow min$.

3.3. **Variation Operators.**

3.3.1. *Recombination.* The standard 2:2 one point crossover operator is used. When used, it will produce either a merging or a splitting of the two clusters involved.

The best two individuals from both parents and offsprings are kept.

3.3.2. *Mutation.* As regarding mutation, special interest has to be paid, as two types are considered.

The first one is in charge of splitting. When a gene is considered for this kind of mutation, a second one that has the same value is searched for, and if a single one found, the current gene will get the number of the next cluster to be formed. Else, nothing happens.

The second type of mutation puts the current instance (given by the index of the current gene) in an existing cluster, whether that instance is or not part of a cluster containing other instances as well. This second mutation operator is clearly in charge either of splitting or merging.

Again the best individual among parent and offspring is accepted in the new population, in both cases.

3.3.3. *Increment.* A new variation operator — *increment* — is introduced. It is applied for every chromosome, taking randomly a gene of the current one, whose value is necessarily zero, and assigning it either the number for the next cluster to be formed or the number of an existing cluster. This operator is in charge of keeping the incremental nature taken over from the IC method. It actually puts an undistributed instance in a new or an existing cluster.

3.3.4. *Stop condition.* The algorithm stops when, after a number of iterations, considered equal in value to the number of the objects in the data set, no progress in the value of the overall fitness function can be observed.

The best chromosome from the final population will give the optimal clustering.

3.4. **Other parameter settings and experimental results.**

Consider a fictional data set that describes the weather conditions for playing some unspecified game[6] given in Table 1.

id	**outlook**	**temperature**	**humidity**	**windy**
x_1	(sunny-0.78,overcast-0.45,rainy-0.20)	(hot-0.90,mild-0.50,cool-0.10)	(high-0.78,normal-0.12)	(true-0.13,false-0.90)
x_2	(sunny-0.80,overcast-0.34,rainy-0.10)	(hot-0.80,mild-0.40,cool-0.20)	(high-0.80,normal-0.20)	(true-0.89,false-0.23)
x_3	(sunny-0.30,overcast-0.85,rainy-0.34)	(hot-0.90,mild-0.30,cool-0.10)	(high-0.90,normal-0.30)	(true-0.16,false-0.77)
x_4	(sunny-0.10,overcast-0.50,rainy-0.90)	(hot-0.40,mild-0.80,cool-0.50)	(high-0.70,normal-0.10)	(true-0.22,false-0.86)
x_5	(sunny-0.13,overcast-0.50,rainy-0.70)	(hot-0.10,mild-0.50,cool-0.80)	(high-0.30,normal-0.80)	(true-0.15,false-0.88)
x_6	(sunny-0.20,overcast-0.40,rainy-0.87)	(hot-0.20,mild-0.40,cool-0.90)	(high-0.30,normal-0.79)	(true-0.77,false-0.30)
x_7	(sunny-0.50,overcast-0.80,rainy-0.30)	(hot-0.30,mild-0.20,cool-0.92)	(high-0.40,normal-0.98)	(true-0.89,false-0.20)
x_8	(sunny-0.90,overcast-0.70,rainy-0.10)	(hot-0.60,mild-0.80,cool-0.20)	(high-0.84,normal-0.22)	(true-0.14,false-0.88)
x_9	(sunny-0.78,overcast-0.34,rainy-0.20)	(hot-0.20,mild-0.60,cool-0.96)	(high-0.13,normal-0.95)	(true-0.10,false-0.98)
x_{10}	(sunny-0.10,overcast-0.50,rainy-0.70)	(hot-0.10,mild-0.90,cool-0.50)	(high-0.24,normal-0.87)	(true-0.34,false-0.68)
x_{11}	(sunny-0.80,overcast-0.30,rainy-0.10)	(hot-0.20,mild-0.87,cool-0.40)	(high-0.32,normal-0.89)	(true-0.56,false-0.45)
x_{12}	(sunny-0.40,overcast-0.90,rainy-0.30)	(hot-0.12,mild-0.90,cool-0.60)	(high-0.82,normal-0.30)	(true-0.85,false-0.30)
x_{13}	(sunny-0.20,overcast-0.90,rainy-0.50)	(hot-0.90,mild-0.50,cool-0.20)	(high-0.40,normal-0.80)	(true-0.65,false-0.22)
x_{14}	(sunny-0.10,overcast-0.30,rainy-0.90)	(hot-0.40,mild-0.78,cool-0.11)	(high-0.98,normal-0.14)	(true-0.94,false-0.12)
	0.2	0.3	0.1	0.4

TABLE 1. The weather data set

We consider the values for the other parameters involved given in Table 2.

population size	**recombination probability**	**mutation probability**
100	0.7	0.7

TABLE 2. Algorithm parameter values

The best chromosome obtained is:

$$5\ 5\ 3\ 6\ 2\ 7\ 7\ 6\ 4\ 2\ 4\ 1\ 8\ 1$$

Therefore, the corresponding classes are:

$$A_1 = \{x_1, x_2\},$$

$$A_2 = \{x_3\},$$

$$A_3 = \{x_4, x_8\},$$

$$A_4 = \{x_5, x_{10}\},$$

$$A_5 = \{x_6, x_7\},$$

$$A_6 = \{x_9, x_{11}\},$$

$$A_7 = \{x_{12}, x_{14}\}$$

and

$$A_8 = \{x_{13}\}.$$

What is very encouraging is that all the final chromosomes provide in 9 cases out of 10 the following clustering results:

(i) instances x_6, x_7 in a cluster,
(ii) instances x_5, x_{10} in a cluster,
(iii) instances x_3, x_8 in a cluster,
(iv) instances x_1, x_2 in a cluster, and
(v) instances x_{12}, x_{14} in a cluster.

3.5. **Conclusions and future work.** The proposed method provides a good enough clustering method. In future work, a better control over the merging and the splitting is desired.

Moreover, a comparison between the results of the original IC method and the evolutionary one would be of real interest.

References

[1] Coello Coello, C., Van Veldhuizen, D., Lamont, G., Evolutionary Algorithms for Solving Multi-Objective Problems, Kluwer Academic/Plenum Publishers, New York, 2002.

[2] Dumitrescu, D., Genetic Algorithms and Evolution Strategies, Blue Publishing House, Cluj-Napoca 2000

[3] Dumitrescu, D., Lazzerini, B., Jain, L., C., Dumitrescu, A., Evolutionary Computation, CRC Press, Boca Raton, Florida, 2000

[4] Gorunescu, R., Evolutionary Incremental Clustering. A New Technique for Detecting Natural Grouping, Research Notes in Artificial Intelligence and Digital Communications, 103, 2003, 73–81

[5] Michalewicz, Z., Genetic Algorithms + Data Structures + Evolution Programs, 2nd edition, Springer - Verlag, 1992

[6] Witten, I., H., Frank, E., Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations, Morgan Kaufmann, 1999

Faculty of Mathematics and Computer Science, Department of Computer Science, University of Craiova, 13 Al. I. Cuza 1100 Craiova Romania

E-mail address: ruxandragorunescu@yahoo.com

Faculty of Mathematics and Computer Science, Department of Computer Science, Babes-Bolyai University, 3400 Cluj - Napoca Romania

E-mail address: ddumitr@cs.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLVIII**, Number 2, 2003

EVOLVING ORTHOGONAL DECISION TREES

JOÓ ANDRÁS AND D. DUMITRESCU

Abstract. Instead of using or fine-tuning the well-known greedy methods to induce decision trees, we propose a new method, which explores the 'brute' force of evolutionary algorithms to evolve decision trees, used mainly for classification. MEP, a new evolutionary technique is used for representing the decision trees.

The paper is organized as follows: the introduction makes a short overview of the decision tree induction techniques. Section 2 contains the short review of the basics of the MEP. Section 3 contains the description of the MEPDTI. Section 4 presents the results, followed by the conclusions, and possible ways to improve the presented method.

Keywords: decision tree, evolutionary algorithms.

1. Introduction

Decision trees are one of the best-known classifiers systems. They classify *instances*, propagating the instance through the decision tree, testing at each internal node one or more *attributes* of the instance. When the instance reaches one of the final nodes of the tree, it is classified: the decision trees have in their leaf nodes the possible *classes* in which the instance can be classified. Decision trees are usually built using a set of *training instances*, called the *training data*, and tested with another set of instances, called the *test data*. The building process is referred as the decision tree *induction*.

Decision trees can be classified using several criteria. One criterion is the *number of classes*: if there are only two classes, they are called binary decision trees. Another criterion is the *type of attributes*, which characterize an instance. There are two main classes of attribute types: *symbolic* (unordered), and *numeric* (ordered). Another axis on which decision trees can be classified is the way in which they are built from the training data (the *tree induction type*). The classical way is a greedy method, on which the most of the tree induction algorithms are based. Another approach is the evolutionary method: the trees are not built, but evolved. But (probably) the main criteria by which the decision trees are categorized is the *test type*, which is made in the internal nodes of the decision tree. There are two

Received by the editors: June 2003.

2000 *Mathematics Subject Classification.* 62C99, 68T20.

1998 *CR Categories and Descriptors.* I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search – *Graph and tree search strategies*; I.5.2 [**Pattern Recognition**]: Design Methodology – *Classifier Design and Evaluation.*

main categories: *univariate* decision trees (also called orthogonal), and *multivariate* decision trees. While univariate decision trees test only one attribute per node, the multivariate ones test more attributes.

Most of the decision tree induction algorithms are based on the following greedy algorithm:

BASIC TREE INDUCTION ALGORITHM

- *If all the training examples at the current node t belong to category C, create a leaf node with the class C.*
- *Otherwise, score each one of the sets of possible splits S, using a quality measure.*
- *Choose the best split s^* as the test at the current node.*
- *Create as many child nodes as there are distinct outcomes of s^*. Label edges between the parent and child nodes with outcomes of s^*, and partition the training data using s^* into the child nodes.*
- *A child node t is said to be pure if all the training samples at t belong to the same class. Repeat the previous steps on all impure child nodes.* [4]

One of the most early, and most famous decision tree induction algorithms is the ID3, invented by Quinlan. ID3 uses the *information gain* as quality measure, to decide at which attribute should split the data. Newer versions of ID3, C4.5, and C5 use the *gain ratio*, to qualify an attribute. Other inducers use other methods to calculate the measure of goodness. For example, the CART algorithm, invented by Briemann, uses the *Gini index*. Other tree induction algorithms include: CN2, a multivariate tree inducer, SPRING and SLIQ developed to handle large datasets, OC1, an oblique tree classifier, and others.

After the tree is built, it is usually too *overfit* to be used in classification. This happens if there are too few representative instances in the training data to 'produce' a true target function, or when there is noise in the training data. Growing the tree in this case still all the training instances are 'consumed' can lead to the effect called overfitting. There are two main methods to combat this situation: to stop the tree growth in the induction phase, or to *prune* back the tree. The second method means elimination of some nodes that seem to not have sufficient evidence.

Another way to build a decision tree is to use an evolutionary method. First an initial tree population is generated randomly. Every individual from the population is evaluated, and some of them are selected and are given the chance to reproduce. In a pure generational model offspring replace the parent generation. The process continues until a termination criterion is satisfied. This is usually connected with the maximum number of generations. Koza [5] was one of the firsts, who applied genetic programming to evolve decision trees. Koza used LISP strings to encode the decision trees into chromosomes. There are direct methods, in which the genetic operators operate right on the decision trees [3].

Evolutionary methods can be combined with other powerful heuristics to produce better results. A successful trial for this is presented in [1].

2. An Alternative Encoding Scheme for Evolutionary Algorithms: Multi-Expression Programming

Multi-Expression Programming (MEP), as introduced in [2], adds an extra step of parallelism to the evolutionary algorithm using a special coding technique. MEP was successfully used in several areas like symbolic regression, evolution of game strategies, or in NP-complete problems like the traveling salesman problem.

MEP uses linear chromosome encoding. Genes of variable length build up every chromosome. Each gene encodes a terminal or a functional symbol. A gene encoding a function contains pointer towards the function arguments. Function parameters are always on a 'lower' level in the chromosome than the function itself. Let us consider the following example:

$1 : a$
$2 : b$
$3 : + \ 1, 2$
$4 : c$
$5 : d$
$6 : + \ 4, 5$
$7 : * \ 3, 6$

This chromosome has seven phenotypic transcriptions, namely:

$E1 : a$
$E2 : b$
$E3 : a + b$
$E4 : c$
$E5 : d$
$E6 : c + d$
$E7 : (a + b) * (c + d)$

Generally each chromosome encodes a number of expressions equal to the number of genes it contains. This is the source of a very strong implicit parallelism.

One might ask which phenotypic transcription should be used when the fitness of a chromosome has to be computed? A possible method is to select the expression, which gives the best fitness. Another way to solve this problem is to let several expressions represent a given chromosome. According to [2] this gives supplementary power to the method.

MEP uses the following evolutionary algorithm:

MEP ALGORITHM

```
begin
    Generate Initial Population;
    $t = 0$;
    Evaluate_Individuals;
    while not Termination_Condition do
```

```
            Elitism;
            Selection;
            Recombination;
            Mutation;
            Evaluate_Individuals;
            endwhile
end
```

For more details regarding MEP, see [2].

3. MEP-Based Decision Tree Induction

An evolutionary classifier system, called MEP-Based Decision Tree Induction (MEPDTI) is proposed. MEPDTI uses the MEP technique to represent the individuals. Classifier systems are used to classify instances from a given data set. Each instance has a set of attributes. Based on the training data set a classifier system builds an internal, and usually compressed representation of the data. This representation is not necessary a decision tree. It may consist from a set of *decision rules*, or some hybrid solution, as in our case.

3.1. MEPDTI Classes.

Instances have to be classified into classes representing meaningful categories. In our model classes are described as simple character strings.

3.2. MEPDTI Attributes.

Each instance is characterized by a set of attributes. In MEPDTI attributes can be of three different forms: *nominal*, *discrete*, and *continuous.*

Nominal attributes are symbolic types, which cannot be ordered. They are characterized with a set of possible values they can take. An example for nominal attribute could be *colour*, with the set of possible values {*red, green, blue*}. Boolean attributes are considered as nominal attributes.

Discrete attributes are characterized by predefined sets of numercal values. An example for a discrete attribute is *age*. This attribute takes values from the set {*0, 1, 2, . . . , 130*}

Continuous attributes are ordered attributes, without a set of predefined values. Optionally a lower and an upper bound can be specified for them. An example for such attribute is *temperature*, which takes values from $(0, +\infty)$ (in Kelvin degrees).

3.3. MEPDTI Instances.

An instance is a vector consisting of possible values of the different attributes, characterizing instances.

Example. Consider the problem of deciding whether a financial organization should or should not offer loan to somebody. The decision relies upon:

- the year income of the applicant (discrete attribute),
- the age of the applicant (discrete attribute),
- criminal records (nominal attribute).

In this case a possible instance representing a person could be:

$$(10000, 47, no).$$

This means, that the applicant has a 10000 income per year, is 47 years old, and has no criminal records.

Usually instances are divided in two categories: *training instances* and *test instances.* While training instances are used to build the decision tree, test instances are used to test the resulting decision trees. Regardless to its type, each instance has associated the correct classification.

3.4. **MEPDTI Representation.** In order to evolve classifiers, the decision mechanism has to be encoded into chromosomes. We use MEP-like chromosomes to represent individuals in MEPDTI. A MEPDTI individual has the following structure:

```
1:        Class_1
2:        Class_2
...
n:        Class_n
n + 1:    if (condition_1^{n+1}) then jump to jp_1^{n+1},
          if (condition_2^{n+1}) then jump to jp_2^{n+1},
          ...,
          else jump to jp_k^{n+1};
...
n + m:    if (condition_1^{n+m} then jump to jp_1^{n+m}
          if (condition_2^{n+m} then jump to jp_2^{n+m},
          ...,
          else jump to jp_1^{n+m};
```

Let us take a closer look to this general form of the MEPDTI chromosome. It is made up by *genes* (a line in the chromosome). In each line, the first number is the gene identification number. This number identifies a gene in the chromosome (but it does not belong to the chromosome). They will be referred as *jump points, entry points* or *nodes.* There are two types of genes: *terminal*, and *non-terminal* genes. Terminal genes are classes and they occupy the first n loci in the chromosome.

Non-final genes are a little bit trickier. They are made up by decision rules. The general form of a decision rule is:

if (condition) then (action).

The *condition* part a tests an instance with respect to an attribute. The general form of a condition is:

(attribute of instance) (operator) (possible value of the attribute)

Depending on the type of the attribute, different operators have been defined:

- for nominal attributes the unique operator is " = ", which test the equality of two nominal values;
- for discrete and continuous attributes the following operators are used: " < " (less), " <= " (less or equal), " > " (greater), " >= " (greater or equal), " = " (equal).

Let us suppose that we want to test the 'year income' attribute at an instance (10000, 47, no). Supposing that the 'year income' attribute takes values from set $\{0, 1, 2, \ldots\}$, a possible condition would be:

$$10000 <= 7329,$$

where 7329 is a possible value of the attribute 'year income', and " <= " is a relational operator. Because this condition evaluates to false, no action is made.

The *action* part of a decision rule, indicates a jump at one of the jump points of the chromosome. The jump point must be smaller than the actual identification number of the gene. This restriction is introduced to avoid recurrent jumps and to obtain only syntactically correct individuals by recombination.

If a condition satisfied a jump is made. But what if the condition is not satisfied? Then the next decision rule from the gene is considered. It could happen that none of the conditions is satisfied. To handle this situation, we introduced an *else* branch, which guaranties that the gene can be leaved in any circumstances.

3.5. **Classification.** The primary job of a chromosome C is to classify instances from different data sets. The classification accuracy of a chromosome is strictly connected whith its fitness. The better a chromosome classifies a data set, the higher its fitness is. For classifying an instance, we start at some 'entry' point of the chromosome, apply the decision rules found there, jump if needed, and continue this until one of the final genes (classes) are reached. The following pseudocode gives the classification algorithm:

CLASSIFICATION ALGORITHM

```
function classify(instance, entry point)
{
        jump to gene denoted by the entry point;
        while (this is not a final gene) do
        {
                take the first rule from gene ;
                while (rule evaluates to false)
                {
                        take next rule;
                }
                if there are no more rules, then jump to node contained by the
'else' branch;
                else jump to node contained by the rule, which evaluated to true;
        }
        return (the class belonging to the final gene)
}
```

Note that due to the MEP structure, the classification highly depends on the chosen entry point. This is because starting from different entry points result in different set of decision rules.

3.6. **Calculating Fitness.** The fitness of an individual in MEPDTI is the measure of how well this individual can classify a given set. So, the fitness is given by the following formula:

$$fitness = \frac{number\ of\ well\ classified\ instances\ from\ set\ S}{total\ number\ of\ instances\ from\ set\ S}$$

An important question arises: where to start the classification, which should be the entry point? To explore the massive parallelism given by the MEP encoding scheme, when the fitness of an individual is calculated ALL possible entry points are used, and the one, which leads to the most correctly classified instances gives the '*number of well classified instances from set S'* used in the above formula.

FITNESS ASSIGNMENT ALGORITHM

```
function fitness(chromosome, set of instances)
{
    max = 0;
    for (every instance x from the set S)
    {
        nr=0;
        for(every entry point ep from the chromosome C)
        {
            if (classify(x, ep)==x.class) then nr++;
        }
        if (max < nr) max = nr;
    }
    return(max/number of instances in the set);
}
```

Remark. *x.class* denotes the correct classification of instance x, as it is given in the training data set.

3.7. **Variation Operators.**

3.7.1. *Recombination.* Recombination ensures the mixture of genes. There are three types of recombination operators in MEPDTI: one-point crossover, two-points crossover, and uniform crossover. All of them affect only the outer part of the chromosome: none of the genes are broken during recombination operations.

3.7.2. *Mutation.* Mutation slightly perturbs a chromosome. It affects only the inner part of the chromosome, (i.e. the genes, and the sub-gene structures). In our implementation the variation operator can take one of more of the following actions:

- change either of operands from a condition
- change the operator from a condition
- change the jump point from a decision rule, or the else branch
- insert or delete decision rules from/into the gene
- change the attribute on which the tests are made from the gene

3.8. **MEPDTI Algorithm.** We are ready now to present the MEPDTI evolutionary algorithm.

MEPDTI ALGORITHM

```
Function MEPDTI(max_epochs)
{
    t = 0;
    Generate a randomly initialized population;
    While (t < max_epochs)
    {
        Calculate the fitness of every individual from the population;
        Sort by fitness the individuals from population;
        While(there are unoccupied positions in the new population)
        {
            Select two individuals from the population, based on their
            fitness using some selection technique;
            Recombine the two selected chromosomes with a specified
            probability, using some recombination technique;
            Apply the mutation operator on the resulting two offsprings
            with a specified probability, using some of the mutation types;
            Put the offsprings into the new population;
        }
        Replace the population with the new one;
        t++;
    }
    Sort by fitness the individuals from population;
    Return (the best individual);
}
```

3.9. **Classification Revisited.** Let us suppose that the evolutionary process ends and we have the best chromosome. One might ask how does this chromosome classify an unknown instance. We reevaluate the chromosome on the training set, and find the entry point, which leads to the most correctly classified instances. This entry point is used then to classify the unknown instance.

3.10. **MEPDTI Complexity and Strong Parallelism.** The strong parallelism hold by MEP has a serious drawback: the evaluation must be done on each possible entry point. This increases the classification complexity of a chromosome (classifying a single instance) in the worst case from $O(n)$ to $O(n^2)$, where n is the number of genes in the given chromosome.

MEPDTI time complexity is $O(m \cdot q \cdot e \cdot n^2)$, where m is the size of the training data, q is the number of chromosomes from the population, and e is the number of generations.

3.11. **Example.** In this section the example of Quinlan [6] is considered. The problem is to decide whether to play or not golf, depending on the weather conditions. There are two classes in this problem {play, don't play}, and four attributes:

- outlook, with possible values: {sunny, overcast, rain}
- temperature, which is continuous
- humidity, also continuous
- windy, with possible values: {true, false}

In the data set there are 14 instances:

```
overcast, 64, 65, true, play
overcast, 72, 90, true, play
overcast, 81, 75, false, play
overcast, 83, 78, false, play
rain, 65, 70, true, don't play
rain, 68, 80, false, play
rain, 70, 96, false, play
rain, 71, 80, true, don't play
rain, 75, 80, false, play
sunny, 69, 70, false, play
sunny, 72, 95, false, don't play
sunny, 75, 70, true, play
sunny, 80, 90, true, don't play
sunny, 85, 85, false, don't play
```

A chromosome evolved by MEPDTI in 250 generations which classifies 100% the above data set is:

```
  0::<<Play>>
  1::<<Don't Play>>
  2::[humidity]::if inst>70.00 then jump to:  0;; if inst>=85.00 then
jump to:  0;; else jump to:  0;;
  3::[windy]::if inst=true then jump to:  1;; else jump to:  0;;
  4::[humidity]::if inst<=70.00 then jump to:  0;; else jump to:  1;;
  5::[outlook]::if inst=sunny then jump to:  4;; if inst=overcast then
jump to:  0;; else jump to:  3;;
```

The best entry point found for test set is: 5. If you look closer to the chromosome, you may see that there is an unused gene, labeled with 2, which can be pruned. After pruning the tree has the following form:

```
0::<<Play>>
1::<<Don't Play>>
```

```
        2::[windy]::if inst=true then jump to:  1;; else jump to:  0;;
        3::[humidity]::if inst<=70.00 then jump to:  0;; else jump to:
1;;
        4::[outlook]::if inst=sunny then jump to:  3;; if inst=overcast
then jump to:  0;; else jump to:  2;;
```

3.12. **Handling Continuous and Missing-Valued Attributes. Pruning.** Handling continuous and missing-valued attributes and tree pruning are important topics in the decision tree induction field. In MEPDTI we used some simple techniques to solve these problems.

Continuous attributes are used without any modification, as they are in the training or test data files. They are mainly used in the random initialization of the chromosomes, and the in the mutation phase.

Before the evolutionary process starts, both data and test files are parsed to eliminate the missing-valued attributes. A simple heuristic is used: if a „?” is found in an instance x (meaning that there is missing value), it is replaced by the most common value from instances which are classified in the same class as x.

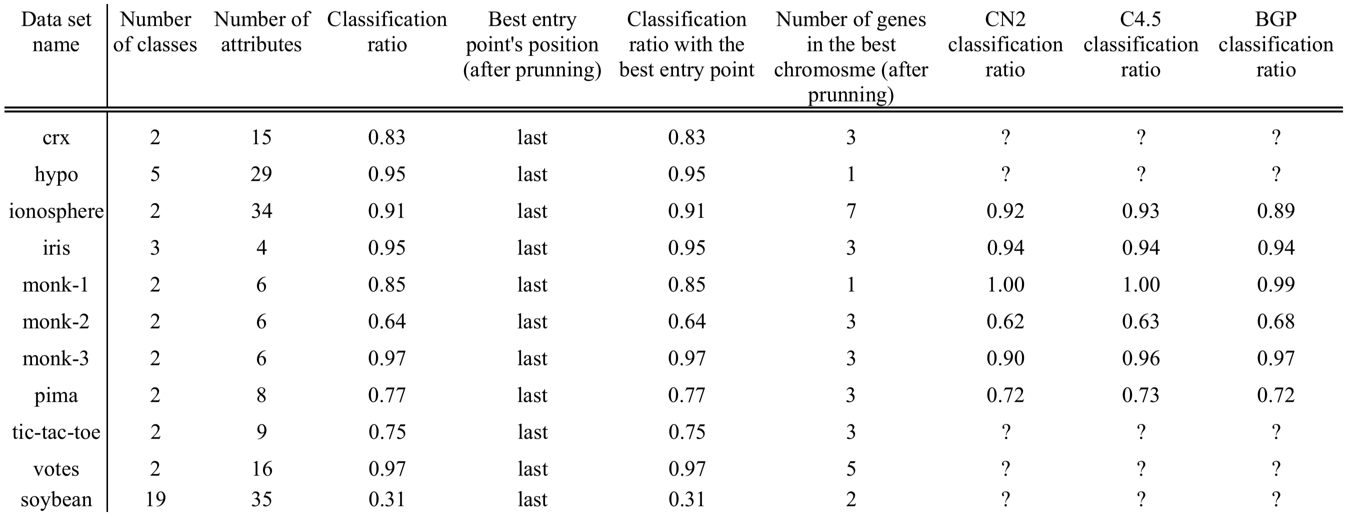
When evolutionary process ends, MEPDTI returns the most successful chromosome, and the most successful entry point of this chromosome. Usually there are some jump points that are not used (starting from the most successful entry point). These corresponding genes are eliminated using a pruning procedure. This procedure parses the chromosome starting at a given entry point, and deletes those genes which are not used in the classification process.

4. Numerical Experiments

MEPDTI produces classifiers whose accuracy is similar with those produced by the classical decision tree inducers, like CN2 or C4.5. For test purposes data from the UCI Repository Of Machine Learning Databases and Domain Theories [7] have been used. The following table presents the results of the MEPDTI. The results of CN2, C4.5, and BGP, an evolutionary method were taken from [3].

The following settings were used:

- population size: 1000
- number of generations: 250
- variable length chromosome usage
- mutation probability: 0.9 (!)
- crossover probability: 0.9
- tournament selection, with tournament size = 4
- no fitness remapping
- two point, or univariate crossover
- strong mutation, meaning that all mutation types described in the previous part of this article are used
- elitism: 1-5%



Data set name	Number of classes	Number of attributes	Classification ratio	Best entry point's position (after prunning)	Classification ratio with the best entry point	Number of genes in the best chromosme (after prunning)	CN2 classification ratio	C4.5 classification ratio	BGP classification ratio
crx	2	15	0.83	last	0.83	3	?	?	?
hypo	5	29	0.95	last	0.95	1	?	?	?
ionosphere	2	34	0.91	last	0.91	7	0.92	0.93	0.89
iris	3	4	0.95	last	0.95	3	0.94	0.94	0.94
monk-1	2	6	0.85	last	0.85	1	1.00	1.00	0.99
monk-2	2	6	0.64	last	0.64	3	0.62	0.63	0.68
monk-3	2	6	0.97	last	0.97	3	0.90	0.96	0.97
pima	2	8	0.77	last	0.77	3	0.72	0.73	0.72
tic-tac-toe	2	9	0.75	last	0.75	3	?	?	?
votes	2	16	0.97	last	0.97	5	?	?	?
soybean	19	35	0.31	last	0.31	2	?	?	?

FIGURE 1. Table showing the results of MEPTDTI

The first remark to the test runs is that although all possible entry points are used for classification, usually the last entry point has the biggest success. However, there were cases when the most successful entry point was an intermediate one.

Another observation is the increased importance of the mutation operator against the crossover. On some data sets the results were the same even if the crossover was given zero probability. The total pass over of the crossover operator needs more investigations.

4.1. **Future Work.** In our approach, only one attribute is tested at each gene. Thus a MEPDTI chromosome corresponds to an orthogonal decision tree. In order to get rid of the drawbacks of univariate decision trees, we intend in the next version to evolve multivariate classifiers.

References

[1] Bala J., Huang J., Vafaie H., DeJong K., Wechsler H., *Hybrid Learning Using Genetic Algorithms and Decision Trees for Pattern Classification.* IJCAI Conference, Montreal, 19-25, 1995.
[2] Oltean M., Dumitrescu D., *Multi Expression Programming*, submitted to Journal of Genetic Programming, 2003.
[3] Rouwhost S.E., Engelbrecht A.P., *Searching the Forest: Using Decision Trees as Building Blocks for Evolutionary Search in Classification Databases.*Proc. Congress on Evolutionary Computation (CEC-2000), 633-638. La Jolla, CA, USA, 2000.
[4] Murthy, S.K., Kasif, S., Salzberg, S., *A System for Induction of Oblique Decision Trees*, Journal of Artificial Intelligence Research 2, 1-32, 1994.
[5] Koza, J., *Genetic Programming: On The Programming of Computers by Means of Natural Selection,* MIT Press, Cambridge, MA, 1992.
[6] Quinlan, R., *C4.5: Programs for Machine Learning*, Morgan Kaufmann, San Meteo, CA, 1993.
[7] UCI Machine Learning Databases and Domain Theories, *ftp.ics.uci.edu: pub/machine-learning-databases*

Department of Computer Science, Babes-Bolyai University, Cluj
E-mail address: jooandras@ms.sapientia.ro
E-mail address: ddumitr@nessie.cs.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLVIII**, Number 2, 2003

A NEW DYNAMIC EVOLUTIONARY CLUSTERING TECHNIQUE. APPLICATION IN DESIGNING RBF NEURAL NETWORK TOPOLOGIES. I. CLUSTERING ALGORITHM

D. DUMITRESCU AND KÁROLY SIMON

Abstract. Recently a new evolutionary optimization metaheuristics, the Genetic Chromodynamics (GC) has been proposed. Based on this metaheuristics a dynamic clustering algorithm (GCDC) is proposed. This method is used for designing RBF neural network topologies. Complexity of these networks can be reduced by clustering the training data. The GCDC technique is able to solve this problem. In Part I the GCDC technique is presented. It is described, how this method could be used for designing optimal RBF neural network topologies. In Part II some numerical experiments are presented.

Keywords and phrases: Dynamic evolutionary clustering, Genetic Chromodynamics, designing neural networks, RBF neural networks.

1. Introduction

By clustering a data set is divided into regions of high similarity, as defined by a distance metric. In most instances, a prototypical vector (the cluster center) identifies a cluster. Hence, the problem of cluster optimization is twofold: optimization of cluster centers and determination of number of clusters. The latter aspect has often been neglected in standard approaches (static clustering methods), as these typically fix the number of clusters *a priori*.

In case of practical problems the number of natural existing clusters is generally unknown. Opposed to static, dynamic clustering does not require *a priori* specification of the number of clusters. Some tentative to develop dynamic evolutionary clustering algorithms are known [5].

Recently a new evolutionary search and optimization metaheuristics - called Genetic Chromodynamics (GC) (see [4, 13]) - has been proposed. Based on this theory a clustering method is proposed. This GC-based dynamic clustering technique (GCDC) can be successfully used for designing RBF neural network topologies.

Received by the editors: September 25, 2003.

2000 *Mathematics Subject Classification.* 68T05, 68T20, 91C20, 92B20.

1998 *CR Categories and Descriptors.* I.2.6 [**Artificial Intelligence**]: Learning – *Connectionism and neural nets*; I.5.3. [**Pattern Recognition**]: Clustering – *Algorithms*.

Solving a problem with a neural network a primordial task is the determination of the network topology. Generally the determination of the neural network topology is a complex problem and cannot be easily solved. When the number of trainable layers and processor units (neurons) is too low, the network is not able to learn the proposed problem. If the number of layers and neurons is too high then the learning process becomes too slow. The main aim is designing optimal topology.

Radial Basis Function (RBF) neural networks are relatively simple neural networks, used especially for solving interpolation problems (see [1, 9, 10, 11, 12, 14, 15, 18]). Complexity of these networks depends on the number of hidden processor units. In the case of the RBF neural networks is dependence between the number of training samples and the number of hidden neurons. Complexity of networks can be reduced by clustering the training data.

Generally, some static clustering techniques are used in order to reduce the complexity of RBF networks. The most popular static clustering algorithms are the k-means type algorithms (see [16, 17]). These methods require *a priori* specification of the number of existing clusters. Dynamic evolutionary clustering techniques could be more efficient for designing optimal RBF neural network topologies (see [6, 7, 8, 19]).

In the next section the GCDC method is described. Section 3 presents how this method can be used for designing RBF neural networks.

2. GC-based Dynamic Clustering

GC [4] is a new kind of evolutionary search and optimization metaheuristics. GC is a metaheuristics for maintaining population diversity and for detecting multiple optima. The main idea of the strategy is to force the formation and maintenance of stable sub-populations.

GC-based methods use a variable-sized population, a stepping-stone search mechanism, a local interaction principle and a new operator for merging very close individuals.

Corresponding to the stepping-stone technique each individual in the population has the possibility to contribute to the next generation and thus to the search progress. Corresponding to the local interaction principle the recombination mate of a given individual is selected within a determined mating region. Only short range interactions between solutions are allowed. Local mate selection is done according to the values of the fitness function. An adaptation mechanism can be used to control the interaction range, so as to support sub-population stabilization. Within this adaptation mechanism the interaction radius of each individual could be different.

To enhance GC, micropopulation models [13] can be used. Corresponding to these models, for each individual a local interaction domain is considered. Individuals within this domain represent a micropopulation. All solutions from a micropopulation are recombined using local tournament selection. When the local domain of an individual is empty the individual is mutated.

Within GC sub-populations co-evolve and eventually converge towards several optima. The number of individuals in the current population usually changes with the generation. A merging operator is used for merging very close individuals. At convergence, the number of sub-populations equals the number of optima. Each final sub-population hopefully contains a single individual representing an optimum, a solution of the problem.

GC allows any data structure suitable for the problem together with any set of meaningful variation/search operators. For instance solutions may be represented as real-component vectors. Moreover the proposed approach is independent of the solution representation.

Based on the GC metaheuristics a new dynamic clustering algorithm - called GCDC - is developed. This technique is described below.

2.1. **Solution Representation.** Let

$$X = \{x_1, ..., x_m\}, x_i \in \mathbf{R}^s, s \geq 1,$$

be the data set for clustering. The cluster structure of X is given by a fuzzy partition $P = \{A_1, ..., A_n\}$ of X. Every class A_i is represented by a prototype $L_i \in \mathbf{R}^s$. $L = \{L_1, ..., L_n\}$ is the representation of the partition P.

In the proposed clustering technique each prototype is encoded into a chromosome. Totality of these chromosomes represents a generation.

The idea of the method is to determine formations of evolving chromosomes converging towards prototypes of real clusters.

The initial population is randomly generated and it contains a large number of individuals. Operations involved in the searching process are selection, crossover, mutation and merging.

2.2. **Fitness Function.** The fitness value of the chromosome L is calculated using the following fitness function:

$$f(L) = \sum_{i=1}^{m} \frac{1}{d^{\alpha}(x_i, L) + C},$$

where $\alpha \geq 1$ and $C > 0$.

The role of the constant C is to prevent infinite or too great values for the fitness function, and together with α controls the granularity of the clusters.

2.3. **Interaction Range.** For each individual in the population (a chromosome representing a prototype) a mating region is considered as the closed ball with radius d^*, where the *interaction radius* d^* depends on the chromosome.

Initially we consider the neighborhood distance for each chromosome as the standard deviation of the all points. For a chromosome L the mean distance $\overline{\delta}$ is calculated between the points in $V(L, d^*)$ and L:

$$\overline{\delta} = \sum_{i=1}^{n_{d^*}} \frac{d(x_i, L)}{n_{d^*}},$$

where $x_1, ..., x_{n_{d^*}}$ are the points in the neighborhood with radius d^* of L.

When the points in $V(L, d^*)$ are uniformly distributed, the value of $\overline{\delta}$ is $\frac{d^*}{\beta}$, where $\beta \in (1, 2]$ is a fixed number, which depends on the dimension s of the search space (generally the best value for β is $\sqrt[s]{2}$). d^* is adjusted such that $\overline{\delta}$ to be equal with $\frac{d^*}{\beta}$, so if $\overline{\delta} \leq \frac{d^*}{\beta}$ then the next value for d^* will be $\beta\overline{\delta}$, else $\overline{\delta}$. If there are not at least two points in the neighborhood of the chromosome, then the previous distance value will be not modified.

2.4. **Genetic Operators.** A micropopulation model is used. At each step of the generation process every chromosome is selected to produce an offspring through crossover or mutation. An individual can be involved into a crossover operation only with individuals that are at smaller distance than d^*. The crossover operation is a convex combination of the codes of the genes. The coefficient of the combination is a randomly generated number for each gene.

The mate for the crossover operation for an individual is selected among the chromosomes in its neighborhood with a proportional selection. Later the mate will be selected as first parent to produce its offspring. For this reason at crossover only one new chromosome is generated. If there is no mate for the crossover operation in the neighborhood of radius d^* of an individual, then the mutation operator will be applied.

Mutation is an additive perturbation of the genes with a randomly chosen value from a $N(0,\sigma)$ normal distribution, where σ is a control parameter called *mutation step size*.

At each generation every chromosome is involved in crossover or mutation. An offspring can replace only its parent. When an offspring is produced, it is compared with the parent and the best (with better fitness) is introduced in the new generation.

An effect of the crossover operation is that the chromosomes in the same subpopulation are overlapping after a number of iterations. When the distance between two chromosomes is smaller than a considered value ε (*merging radius*) they are merged. In this way the size of the population decreases during the process until n individuals remain, where n is the optimal cluster number.

2.5. **Termination and Fuzzy Class Detection.** If no more changes occur in the population through a fixed number of iterations then the process will stop. Individuals constituting the last population are considered as prototypes of the detected clusters. For all data points the fuzzy membership degrees to the clusters determined by the prototypes are calculated.

3. Designing RBF Neural Networks Using the GCDC Technique

Complexity of RBF neural networks depends on the number of hidden processor units. There is dependence between this number and the number of training samples. Complexity of networks can be reduced by clustering the training data.

3.1. **Designing and Training RBF Neural Networks.** RBF is a feed-forward neural network with an input layer (made up of source nodes: sensory units), a single hidden layer and an output layer. The network is designed to perform a nonlinear mapping from the input space to the hidden space, followed by a linear mapping from the hidden space to the output space.

The activation functions for the processor units in the hidden layer are radial basis functions (for example Gaussian functions). These functions generally have two parameters: the center and the width. The argument of the activation function of each hidden unit computes a distance between the input vector and the center of that unit.

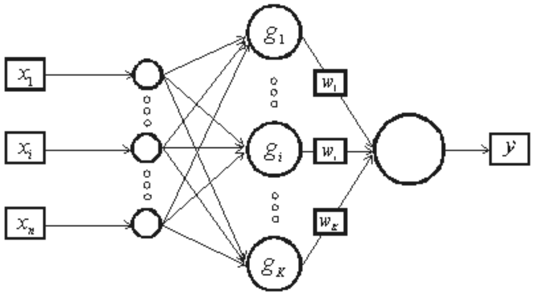


Figure 1. RBF neural network topology

If $\mathbf{x}= (x_1, ..., x_n)$ is the input vector, $g_i()$ is the activation function and w_i is the synaptic weight corresponding to the i^{th} hidden neuron, then the output created by the network will be:

$$y = \sum_{i=1}^{K} w_i g_i(\mathbf{x}).$$

Usually Gaussian functions are used as RBF. In this case we may consider:

$$g_i(\mathbf{x}) = \mathbf{e}^{-\frac{\|\mathbf{x}-\mathbf{c_i}\|^2}{2\sigma_i}},$$

where c_i is the center parameter and σ_i is the variance (width parameter) for the function corresponding to neuron i.

The hidden layer of the RBF neural networks may be trained with a supervised learning algorithm. The aim is to establish the synaptic weights of the network. A descendent gradient-based algorithm can be considered.

Let us to note:

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}, i = 1, ..., K,$$

where η is the *learning rate* and E is the *global learning error*.

At the l^{th} step of the learning process the global learning error is calculated according to the formula:

$$E_l = \frac{1}{N} \sum_{i=1}^{N} (z_i - y_i)^2$$

where N is the number of points in the training data set, z_i is the desired output and y_i is the network output.

Network weights are modified according to the following correction rule:

$$w_i = w_i + \Delta w_i, i = 1, ..., K.$$

3.2. **Using GCDC for Designing Optimal RBF Neural Network Topologies.** Complexity of RBF neural networks depends on the number of hidden neurons. This is the number of radial basis functions with different center parameters. It should be favorable the use of training samples as RBF centers, but in some cases this is impossible. If few training points are present then all of them should be used as centers. In this case the number of processor units in the hidden layer is equal with the number of training samples. If the number of training samples is high, then not all of them might be used (the number of hidden processor units must be reduced). In this situation the locations of the centers may be chosen randomly from the training data set. In practical situations this solution is not very efficient. A better idea is to consider a single neuron for a group of similar training points. These groups of similar training points can be identified using clustering methods.

Generally, some static clustering techniques are used in order to solve the RBF center detection problem. GCDC does not require *a priori* specification of the number of clusters. The algorithm is able to determinate this number, so it can be used for designing optimal RBF neural network topologies.

Let

$$T = \{(x_i, z_i) | x_i \in R^n, z_i \in R, i = 1, ..., N\},$$

be the set of training samples.

The GCDC algorithm is used for clustering this training set. A RBF neural network is considered. The number of neurons in the hidden layer of the network is K, where K is the number of clusters determined by the GCDC method. Cluster centers identified by the GCDC algorithm are used as center parameters for the activation functions. Width parameters can be determined corresponding to the diameter of the clusters. In this way optimal RBF neural network topology can be obtained.

4. Conclusions

Within clustering problems a primordial task is the determination of the number of natural existing clusters. Dynamic clustering techniques are able to solve this problem.

Based on the GC metaheuristics, GCDC is a new evolutionary technique for dynamic clustering. The method can be used for designing optimal RBF neural network topologies.

References

[1] Broomhead D. S., Lowe D.; *Multivariable Functional Interpolation and Adaptive Networks*, Complex Systems, 2 (1988), pp. 321-355.

[2] Dumitrescu D.; *Algoritmi Genetici şi Strategii Evolutive - Aplicaţii în Inteligenţa Artificială şi în Domenii Conexe*, Editura Albastra, Cluj Napoca, 2000.

[3] Dumitrescu D., Lazzerini B., Jain L. C., Dumitrescu A.; *Evolutionary Computation*, CRC Press, Boca Raton, 2000.

[4] Dumitrescu D.; *Genetic Chromodynamics*, Studia Univ. Babes-Bolyai, Ser. Informatica, 35 (2000), pp. 39-50.

[5] Dumitrescu D.; *A New Evolutionary Method and its Applications in Clustering*, Babeş-Bolyai University, Seminar on Computer Science,2 (1998), pp. 127-134.

[6] Dumitrscu D., Simon K.; *Evolutionary Clustering Techniques for Designing RBF Neural Networks*, Babeş-Bolyai University, Seminar on Computer Science, (2003).

[7] Dumitrscu D., Simon K.; *Reducing Complexity of RBF Neural Networks by Dynamic Evolutionary Clustering Techniques*, Proceedings of the 11^th Conference on Applied and Industrial Mathematics, (2003).

[8] Dumitrescu D., Simon K.; *Genetic Chromodynamics for Designing RBF Neural Networks*, Proceedings of SYNASC, (2003).

[9] Enăchescu C.; *Caracterizarea Reţelelor Neuronale ca şi Metode de Aproximare- Interpolare*, Petru Maior University, Buletinul Stiintific, 7 (1994).

[10] Enăchescu C.; *Elemente de Inteligenţă Artificială*, Petru Maior University, Tg. Mureş, 1997.

[11] Haykin S.; *Neural Networks*, Macmillan College Publishing Company, New York, 1994.

[12] Moody J., Darken C.; *Fast Learning in Networks of Locally Tuned Processing Units*, Neural Computation, 1 (1989), pp. 281-294.

[13] Oltean M., Groşan C.; *Genetic Chromodynamics Evolving Micropopulations*, Studia Univ. Babes-Bolyai, Ser. Informatica, (2000).

[14] Poggio T., Girosi F.; *Networs for Approximation and Learning*, Proceedings of IEEE, 78 (1990), pp. 1481-1497.

[15] Powell M. J. D.; *Radial Basis Functions for Multivariable Interpolation: A review*, in Algorithms for Approximation, J. C. Mason and M. G. Cox, ed., Clarendon Press, Oxford, 1987, pp. 143-167.

[16] Schreiber T.; *A Voronoi Diagram Based Adaptive k-means Type Clustering Algorithm for Multidimensional Weighted Data*, Universitat Kaiserslautern, Technical Report, (1989).

[17] Selim S. Z., Ismail M. A.; *k-means Type Algorithms: A Generalized Convergence Theorem and Characterization of Local Optimality*, IEEE Tran. Pattern Anal. Mach. Intelligence, PAMI-6, 1 (1986), pp. 81-87.

[18] Simon K.; *OOP Pentru Calculul Neuronal*, Petru Maior University, Dipl. Thesis, 2002.

[19] Simon K.; *Evolutionary Clustering for Designing RBF Neural Networks*, Babeş-Bolyai University, MSc. Thesis, 2003.

Babeş-Bolyai University, Faculty of Mathematics and Computer Science, Department of Computer Science, Cluj-Napoca, Romania
E-mail address: `ddumitr@cs.ubbcluj.ro`

E-mail address: `ksimon9@yahoo.com`

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLVIII**, Number 2, 2003

COORDINATION AND REORGANIZATION IN MULTI-AGENTS SYSTEMS, I

ALINA BACIU AND ADINA NAGY

Abstract. A method of considering coordination and reorganization as keys in achieving (organizational) multi-agent system adaptation in unknown situations is proposed. Within a not totally predictable environment multi-agent systems are prone to failures. In such unpredicted situations the system must be able to adapt in order to accomplish its purpose.

The proposed system architecture is a combination of MOISE+ and MOCA concepts. The main inconvenient of MOCA platform is that the mechanism of dynamic role allocation is entirely left to the designer. The inconvenient of MOISE+ platform is that agents' behavior is not considered.

MOCA gives a structural vision on multi-agents systems based on individual and collective patterns of behavior. MOISE+ model describes how agents endorse roles in order to achieve their individual and collective goals.

In Part I the main concepts of MOCA and MOISE+ models are presented.

Keywords: Additional Key Words: Multiagent Systems Reorganization, Role Endorsement Mechanism

1. Introduction

The term agent is difficult to define. The main point about agents is they are autonomous: capable of acting independently, exhibiting control over their internal state. Thus: an agent is a computer system capable of autonomous action in some environment. An intelligent agent is a computer system capable of flexible autonomous action in some environment. By flexible, we mean (Wooldridge and Jennings, 1995) *autonomy, reactivity, pro-activeness* and *social ability.*

1.1. **Multi Agent Systems.** A Multi Agent System (MAS) is a network of autonomous entities (agents) that work together in order to achieve a global goal, the system goal. Data in the system is decentralized and there is no agent that can accomplish by himself the system's goal, meaning that agents need each other to

Received by the editors: October 8, 2003.

2000 *Mathematics Subject Classification.* 68T05.

1998 *CR Categories and Descriptors.* I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence – *Multiagent systems, Coherence and coordination.*

achieve the system goal. A MAS has two main properties, which seems controversial: the global purpose that must be achieved and the autonomous agents. While the autonomy of the agents is essential for the MAS, it may also cause the looseness of the global coherence. In these conditions MAS organization is used to solve this conflict, constraining agent's behavior towards the system global purpose.

1.2. **Mainstream researches in MAS.** Mainstream researches in MAS correspond to the MAS central concepts: namely that of *agent* and *group*. The agent-centered approaches are concerned on how to represent agent 'internal' knowledge (such as *Believes, Intentions* and *Desires*, in a so-called *mentalist* approach) or internal behavior, as well as their local interactions and environment (Nagy, 2002).

In the framework of organizational systems, three dimensions are used to describe the MAS:

- the structure expressed through roles and groups,
- the functioning (global plans and tasks) and
- deontic relations or other norms (agents' obligations, norms, responsibilities, permissions etc.).

Agent and organization based approaches share the same goals: explain what a multi-agent system is, how it works and how it can be used. The main difference is the set of basic concepts. While both of them allow for a sociological dimension - local interactions for the agent-centered approach – the organization-centered approach has a real social dimension.

When describing an organization, one of the encountered problems is to define these aspects in such a way that they can be assembled in a single coherent specification. The existent models of MAS concentrate mainly on a single dimension. Such is the MOCA platform (Amiguet, Müller, Báez, and Nagy, 2002) and the Agent-Group-Role (AGR) models or others concentrated on the deontic aspect of the system (Barbuceanu and Lo, 2000).

Our research is located in the field of organization-centered systems where agents are able to represent the organization they evolve in. The proposed model combines the existent concepts in two models: MOCA model and MOISE+ model.

2. MOCA platform

There are mainly two types of approaches of organizational multi-agent systems in literature: some systems allow for dynamic social organization but social structures do not impose anything on agent's behavior; other models do consider social structures as recurrent patterns of interaction, but then the social structures are usually static.

The contribution of MOCA (Model of Organization Centered on Agents) platform is to combine these two approaches by allowing the designer to describe the organizations with their roles, relationships and dynamics (protocols), but simultaneously allowing any agent to dynamically create a group instantiating an organization, to enter and to leave such a group.

MOCA is the first realized platform that gives both the theoretical background and an operational semantics for the notion of (*behavioral) role* and *organization.* In what follows, we will distinguish MOCA's concepts of role and organization by mentioning the platform *MOCA-role*, *MOCA-organizational structure*, etc.

While the deontic and other specifications of multi-agent systems in terms of agents' obligations, norms, responsibilities, permissions, etc. can be classified as *mentalist* (Parunak and Odell, 2001) approaches to organizational design, the specification of MOCA organization belongs to the class of *behaviorist* approaches (see Nagy, 2003).

A MOCA-organization is a recurrent pattern of interaction, representing a global, but specific (meaning several view-points on a system can co-exist and interact) view on a system. Such an organization is represented by a graph where the vertices are MOCA-role descriptions and the edges are inter-role relations, which specify – among other properties – the types of perturbations a role can generate and receive.

A *MOCA-role* is an individual recurrent pattern of behavior, within an organization. MOCA-roles are specified through the mechanism of state-charts, which identify the sequences of state-transition and actions firing, according to internal conditions and perturbations received by the agent endorsing a role.

MOCA leaves a high autonomy to agents, which can endorse and leave a role according to their individual goals (this part is left for a further work and it is here that our model enriches MOCA). However, the role endorsement is driven by agent competences.

A MOCA-role specifies what competences an agent should have in order to endorse it. Also some competences can be provided to the agent through the roles the agent endorses. The agent's competences are expressed as components of the role, and implemented, on the MOCA platform, as java interfaces.

MOCA notions are structured on two axes: a two levels structure and an internal-external distinction (see Figure 1).

The distinction between the executive level and the descriptive level is the same as the distinction between the class and the object in the object oriented programming. We can tell that the executive level is the instantiation of the descriptive level.

The internal - external distinction related to the agent shows the role position in this model: the role realizes the link between the internal state of the agent and the system he evolves in.

The main inconvenient of MOCA is that it leaves the system designer without any tool for assigning roles to agents. Another shortcoming can be the management group. This group is responsible for group's dynamic (agents entering and leaving

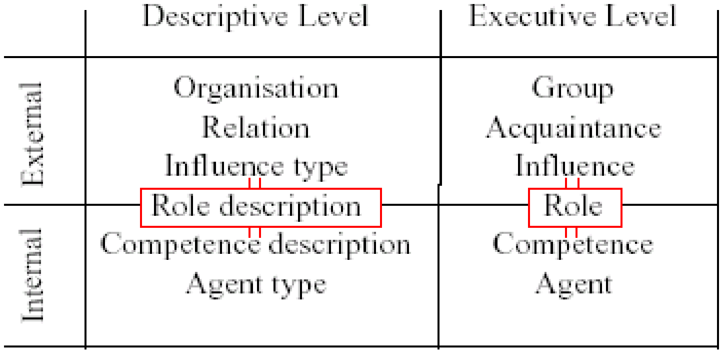


FIGURE 1. The main concepts of MOCA

groups). This can be a bottleneck point in the system and its failure could lead to system failure.

3. MOISE+ MODEL

A new direction in MAS is to join the organizational "roles" with global and individual plans. In MOISE model three levels were identified:

- *individual level* - representing the behaviors that an agent is responsible for when it adopts a role,
- *social level* - describing interconnections between roles and
- *collective level* - that represents the roles aggregation in large structures.

The organization-centered models usually concern only one direction of the two: the system functioning, meaning system's global plans or the system structure. Although MOISE tries to concern about these two aspects, its main shortcomings (for reorganization) are the lack of the concept of an explicit global plan and the strong dependence among the structure and the functioning.

MOISE+ is a model conceived to assemble the three levels of the system in a coherent MAS organizational description, suitable for the reorganization process. This was accomplished by specifying the first two dimensions almost independently of each other and after properly linking them by the deontic dimension (Hübner, Sichman, Boissier, 2002).

Figure 2 shows how system global purpose can be achieved by constraining agents' behavior by organization's structure but also providing them some tested plans for goal achievement through organization functional specification. In this way agents have some tested plans to follow, they have the choice of reasoning for a plan to work together when there is no plan to be followed and this must not be done each time they want to work together, because experience is stored in system knowledge base.

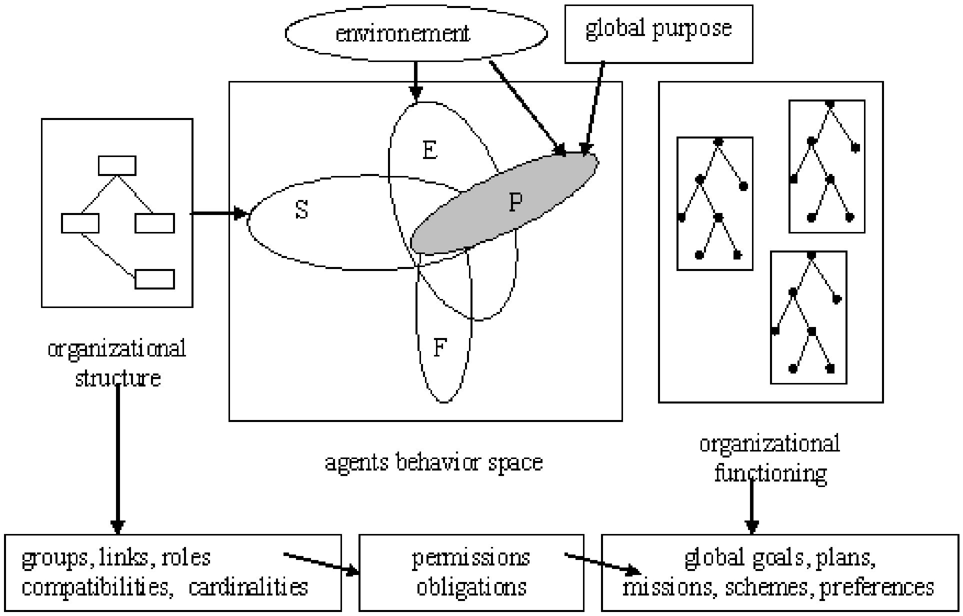

environement
global purpose
E
S
P
F
organizational structure
agents behavior space
organizational functioning
groups, links, roles compatibilities, cardinalities
permissions obligations
global goals, plans, missions, schemes, preferences

FIGURE 2. The organization effects on MAS

Within MOISE+ platform three dimensions can de distinguished:

- structural dimension,
- functional dimension,
- deontic dimension.

3.1. **Structural dimension.** Within MOISE + three main concepts - roles, role relations and groups - are used to build the individual, social, and collective structural levels of an organization.

The *individual level* is formed by the roles of the organization. A *role* means a set of constraints that an agent ought to follow when it accepts to enter a group playing that role. The constraints imposed by the role are defined in relation to other roles (in the collective structural level) and in relation to global plans.

The *social level* is used to specify relations between roles, relations that directly constraint the agents. The relations between roles are called *links* and are represented like predicates: link(s, d, t) where s is the link source, d is the link destination and t is the link type that can be authority, communication or acquaintance.

These links are used to constrain agents after they have accepted to play a role.

The *collective level* imposes some constraints regarding the roles an agent can play at the same time. The roles can be played only in a group already created. A *group* is an instantiation of a group specification.

When specifying a group the following elements must be stated:

- roles that may be played in the group;
- sub groups the group has;
- links between the roles in the group;
- links that can exist between agents playing a role in the group and agents playing different roles in other groups;
- compatibilities between the roles played by the same agent in the same group and also in other groups;
- for each role- the maximum and minimum number of agents that can play a role in the group in order it to be well formed;
- for each group - the minimum and the maximum number of subgroups that can be created.

3.2. **Functional dimension.** The functional dimension describes how the global goals are decomposed by plans and distributed to the agents by missions**.** At the collective level, this means that there is a global plan decomposed in *schema* and at the individual level there are *missions* that an agent may be committed to.

Each goal can be decomposed in sub-goals through plans, which may use three operators: sequence (the sub-goals must be achieved one after the other), choice (only one of sub-goals must be achieved) and parallelism (the sub-goals can be achieved in parallel). At this level there is also an order for missions telling in a given situation the success rate of each mission. Using this order the agents may choose the mission that looks to be the most promising for the global goal achievement.

3.3. **Deontic dimension.** *Deontic dimension* relates the structure and the functioning dimensions describing the permissions and obligations from roles to missions at specified moments in time. The deontic dimension is thus a set of obligations and permissions for the agents playing different roles on schema decomposed in missions.

A MOCA organizational system may allow for agents without any internal content other than the ability to send and receive messages, to enter organization (an agent enters an organization by endorsing a role in that given organization). In MOISE+ nothing is said about the internal abilities or competences of agents and the present architecture in not self-consistent.

Furthermore, it is beyond the scope of the quoted papers on MOISE+ to describe the instantiation process, and particularly the correspondences between the competences of agents and the behaviors specified by the MOISE+ roles. This correspondence between role competences and agents arises naturally within MOCA.

Therefor we consider that MOCA and MOISE+ could be seen as complementary approaches. MOCA gives a good operational semantics to the notions of organizational structure, organization and role, together with a fully operational platform. Notion of competence required from an agent to endorse a role is clearly defined and the conditions where an agent is eligible to endorse a role are put.

MOISE+ describes the constraints for the role endorsement by agents, as well as constraints about the cardinality of the final multi-agent system in order to achieve the global goal of the organization. Another advantage is that conditions for role compatibility within the same agent can be given because MOISE+ allows specifying for every role, if it can be endorsed or not by an agent having other given roles.

4. Conclusions

This paper presented the models MOCA and MOISE+ which are the basis for a new multi-agent system model. This new model will be introduced in Part II.

References

[1] Amiguet, M., Müller, J-P., Báez, J, Nagy, A. 2002. The MOCA Platform: Simulating the Dynamics of Social Networks, Workshop of Multi-Agent–based simulation, MABS'02, Barcelona

[2] Amiguet, M. 2003. MOCA: un modèle componentiel dynamique pour les systèmes multi-agents organisationnels, thèse de doctorat, Université de Neuchâtel (Switzerland)

[3] Baez, J. 2002. Extension et consolidation de la plate-forme organisationnelle MOCA, mémoire de diplôme, Université de Neuchâtel (Switzerland)

[4] Barbuceanu, M., Lo, Wai-Kai. 2000. Integrating individual and social reasoning models for organizational agents

[5] Ferber, J. 1999. Multi-Agent Systems, Addison Wesley

[6] Hannoun, M., Boissier, O., Sichman, J.S., Sayettat, C..1999. Moise : Un modèle organisationnel pour la conception de systèmes multi-agents, Editions Hermés

[7] Hübner, J.F., Sichman J.S., Boissier O. 2002 A Model for the Structural, Functional, and Deontic Specification of Organizations in Multiagent Systems

[8] Olivier Gutknecht, O., Ferber, J. 2002. MadKit v2.0.1, LIRMM, Université Montpellier II, 2000, http://www.madkit.org

[9] Parunak, H., Odell, J. 2001.Representing social structures in UML, AOSE2001

[10] Odell, J., Parunak, H., Fleischer, M. 2003. The Role of Roles in Designing Effective Agent Organizations, *Software Engineering for Large-Scale Multi-Agent Systems*, Alessandro Garcia et al, LNCS, Springer

[11] Kumar, S., Cohen, P. R., Levesque, H. 2000. The Adaptive Agent Architecture: Achieving Fault-Tolerance Using Persistent Broker Teams, Fourth International Conference on Multi-Agent Systems (ICMAS-2000), Boston

[12] Mellouli, S., Mineau, G., Pascot, D. 2002. Multi-Agent System Design, ESAW'02, Engineering Societies in the Agents World, Madrid, Spain

[13] Nagy, A. 2002. Behaviorist organizational models for the MAS – a state of the art, Technical Report, University of Neuchâtel, Switzerland

[14] Glasser, N., Morignot, P. 1997. The Reorganization of Societies of Autonomous Agents

Faculty of Mathematics and Computer Science, Babes-Bolyai University of Cluj-Napoca, Romania
E-mail address: alina_baciu@yahoo.com

Institute of Computer Science and Artificial Intelligence, University of Neuchâtel, Switzerland
E-mail address: adina.nagy@unine.ch

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLVIII**, Number 2, 2003

HIERARCHICAL CLUSTERING ALGORITHMS FOR REPETITIVE SIMILARITY VALUES

DANA AVRAM LUPŞA, GABRIELA ŞERBAN, AND DOINA TĂTAR

Abstract. This paper presents a novel variant of the hierarchical clustering from [2]. We tried to solve the problem of repetitive similarity values that appears on distributional similarity values. Also we propose an algorithm to build a similarity tree as a taxonomy that respects the hierarchical clusters determined above.

1. Introduction

Bootstrapping semantics from text is one of the greatest challenges in natural language learning. Clustering nouns can be useful in construction of a set of synonyms for word sense disambiguation, to perform query expansion in QA systems [9], to build ontology from a text, in data mining, etc., especially for languages others than English, for which doesn't exist a hierarchy such as WordNet (as in Romanian language case). One very surprising approach is an unsupervised algorithm that automatically discovers word senses from text.

Automatic word sense discovery has applications of many kinds. It can greatly facilitate a lexicographer's work and can be used to automatically construct corpus-based similarity trees or to tune existing ones.

We study distributional similarity measures for the purpose of improving some noun clustering methods [2]. We suggest two algorithms that obtain clusters and similarity trees for nouns. Starting with hierarchical clustering algorithm, we consider the case when the similarity values can repeat and suggest a method to determine the taxonomy with respect of hierarchical clusters found by the hierarchical clustering algorithm.

This paper is organized as follows. In section 2, we present some methods that extract words similarity from untagged corpus. A comparison among the precision of the results is also made. Section 3 describes the agglomerative algorithm for hierarchical clustering and it's modified version. Some experimental results are also shown. In section 4, we present the novel agglomerative algorithm for similarity tree. We outline the similarity between the clustering algorithm and the similarity

Received by the editors: October 15, 2003.

2000 *Mathematics Subject Classification.* 62H3, 68Q25, 65Q55, 68R10, 68T50.

1998 *CR Categories and Descriptors.* I.2.7 [**Artificial Intelligence**]: Natural Language Processing – *Text analysis* ; I.5.4. [**Pattern Recognition**]: Applications – *Text processing* .

tree for the experimental results considered. Finally, section 5 sketches applications of the algorithm and discusses future work.

2. Word similarities

Semantic knowledge is increasingly important in NLP. The key of organizing semantic knowledge is to define reasonable similarity measures between words. In many papers the similarity between two words is obtained by the n-grams models [11], by mutual information [3] or by syntactic relations [13]. One other way to define this similarity is the vector space model [5, 12, 7] which we use in this paper. The idea of vector-based semantic analysis is to understand the meaning of a word one has to considering its use in the context of concrete language behavior. The distributional pattern of a word is defined by the contexts in which the word occurs, where context is defined simply as an arbitrarily large sample of linguistic data that contains the word in question.

Syntactic analysis provides some potentially relevant information for clustering [10]. For a corpus in Romanian language the relation predicate-object or subject-predicate can be estimated after position: the object is almost always after the predicate, the subject is before the predicate. So we replaced a syntactical analysis by constructing context vectors as in **Definition 2**.

The reason for using narrow context windows as opposed to arbitrarily contexts is the assumptions that the semantically most significant context is the immediate vicinity of a word. That is, one would expect the words closest to the focus word to be of greater importance than the other words in the text.

Definition 1. *In* **AlgUnord** *algorithm* ([2]) *the vector*
$\vec{w_i} = (w_i^1, w_i^2, \cdots, w_i^m)$
is associated with a noun w_i *as following: let us consider that* $\{v_1, v_2, \cdots, v_m\}$ *are* m *verbs of a highest frequency in corpus. We define:*

$$w_i^j = \textit{number of occurences of the verb } v_j \textit{ in the same context with } w_i$$

Let us remark that other vector-space models were used in the literature. For example, in [1] is presented a hierarchy of nouns such that the vector $\vec{w_i} = (w_i^1, w_i^2, \cdots, w_i^m)$ associated with a noun w_i is constructed as follows: $w_i^j = 1$, if the noun w_j occurs after w_i separated by the conjunction *and* or an appositive, or else $w_i^j = 0$.

Definition 2. *In* **AlgOrd** *algorithm* ([2, 5]) *the vector* $\vec{w_i}$ *is associated with a noun* w_i *as following: for each verb* v_j *is calculated a sub-vector* $(v_j^{-3}, v_j^{-2}, v_j^{-1}, v_j^{+1}, v_j^{+2}, v_j^{+3})$ *where* v_j^{-3} *=1 if* v_j *occurs in a windows context of* w_i *in the position -3 or* v_j^{-3} *=0 else, and so far for* $v_j^{-2}, v_j^{-1}, v_j^{+1}, v_j^{+2}, v_j^{+3}$.

Finally, the vector $\vec{w_i}$ *is obtained by the concatenation, in order, of all sub-vectors of verbs* $\{v_1, v_2, \cdots, v_m\}$.

Let us remark that in **AlgOrd** the number of components of the noun's vector $\vec{w_i}$ is $6 \times m$, while in **AlgUnord** is m. The dimension of a window can be 4 (so the subvectors for a verb v_j are $v_j^{-2}, v_j^{-1}, v_j^{+1}, v_j^{+2}$) or 2 (and the subvectors are: v_j^{-1}, v_j^{+1}). We will denote the windows in each case by 3+3, 2+2 or 1+1.

In both algorithms, if a noun w_i occurs in more contexts, the final vector $\vec{w_i}$ is obtained as the average of all the context vectors.

Let us observe that the corpus does not have to be POS tagged or parsed and that one can use a stemmer to recognize the flexional occurrences of the same word (Romanian language is a very inflexional language).

Let us consider that the objects to be clustered are the vectors of n nouns, $\{w_1, w_2, \cdots, w_n\}$ and that a vector is associated with a noun w_i as above.

The similarity measure between two nouns w_a, w_b is the *cosine* between the vectors $\vec{w_a}$ and $\vec{w_b}$ [6]:

$$cos(\vec{w_a}, \vec{w_b}) = \frac{\sum_{j=1}^{m} w_a^j \times w_b^j}{\sqrt{\sum_{j=1}^{m} w_a^{j^2}} \times \sqrt{\sum_{j=1}^{m} w_b^{j^2}}}$$

and the distance (dissimilarity) is $d(\vec{w_a}, \vec{w_b}) = \frac{1}{cos(\vec{w_a}, \vec{w_b})}$.

In **Table 1** we present, comparatively, the precision of the clustering algorithms for our clustering experiment.

	AlgOrd (3+3)	AlgUnord
non-hierarchical	63%	54%
hierarchical	45%	36%

TABLE 1. Precision of clustering algorithms for the proposed experiment

In the followings, we will consider the results of the studied hierarchical algorithms (see Table 1). The decision was made to support the study of repetitive similarity values. The similarity values are repetitive more significant for the hierarchical algorithm than for the non-hierarchical ones.

The distributional similarity matrices obtained for the Romanian words: *asociatie, durata, localitate, oameni, oras, organizatie, partid, persoana, perioada, sat, timp* by the considered hierarchical algorithms are presented in **Table 2** and **Table3**. For readability reasons the values shown are rounded to 9 decimal characters.

The similarity values are repetitive, as shown in the **Fig 1**.

In what follows we will give an algorithm for hierarchical clustering, that handle repetitive values.

3. New hierarchical clustering algorithm

Word clustering is a technique for partitioning sets of words into subsets of semantically similar words and is increasingly becoming a major technique used in

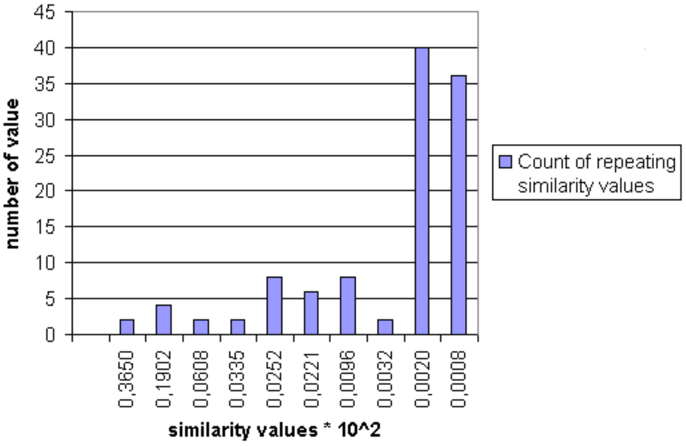


FIGURE 1. Repetitive similarity values obtained by hierarhical algorithm **AlgUnord**

a number of NLP tasks ranging from word sense or structural disambiguation to information retrieval and filtering. In the literature [4], two main different types of similarity have been used. They can be characterized as follows:

1. *paradigmatic or substitutional similarity*: two words that are paradigmatically similar may be substituted one for another in a particular context. For example, in the context *I read the book* , the word *book* can be replaced by *magazine* with no violation of the semantic well-formedness of the sentence, and therefore the two words can be said to be paradigmatically similar;

2. *syntagmatic similarity*: two words that are syntagmatically similar significantly occur together in text. For instance, *cut* and *knife* are syntagmatically similar since they typically co-occur within the same context.

Both types of similarity, computed through different methods, are used in the framework of a wide range of NLP applications.

The agglomerative algorithm for hierarchical clustering that we intend to use is part of the second category. The original hierarchical clustering algorithm [2, 6] is described in what follows.

Agglomerative algorithm for hierarchical clustering

Input

The set $X = \{w_1, w_2, \ldots, w_n\}$ of n words to be clusterised,
the similarity function $sim : X \times X \to R$.

Output

The set of hierarchical clusters

```
    $C = \{C_1^0, C_2^0, \ldots, C_j^n\}$

BEGIN
    FOR $i := 1$ TO $n$ DO
        $C_i^0 := w_i$
    ENDFOR
    $step := 0$
    $C^0 := \{C_1^0, C_2^0, \ldots, C_n^0\}$
    $C := C^0$
    WHILE $|C| > 1$ DO
        $step := step + 1$
        $C^{<step>} := C^{<step-1>}$
        $(C_{u*}^{<step>}, C_{v*}^{<step>}) :=$
                $argmax_{(C_u^{<step>}, C_v^{<step>})} sim(C_u^{<step>}, C_v^{<step>}),\ u <> v$
        $C_*^{<step>} := C_{u*}^{<step>} \cup C_{v*}^{<step>}$
        $C^{<step>} := (C^{<step>} \setminus \{C_{u*}^{<step>}, C_{v*}^{<step>}\}) \cup C_*^{<step>}$
        $C := C \cup C^{<step>}$
    ENDWHILE
END
```

As similarity $sim(C_u, C_v)$ we considered *average -link* similarity:

$$sim(C_u, C_v) = \frac{\sum_{a_i \in C_u} \sum_{b_j \in C_v} sim(a_i, b_j)}{\mid C_u \mid \times \mid C_v \mid}.$$

Taken as input the similarities from **Table 2**, the resulting hierarchical clusters are shown in **Fig 2** . The circles indicate the clusters at a certain moment and the numbers indicate the step when the cluster was formed.

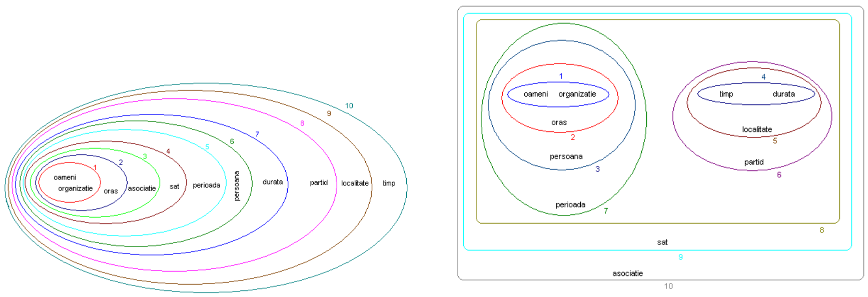


FIGURE 2. Results of agglomerative algorithm for hierarchical clustering on experimental data set (table 2 and 3)

When the similarity values have many repetitive values, as shown in **Fig 1**, it could be possible that the similarity between different clusters is the same. The idea behind the new hierarchical clustering algorithm is to consider at each step all the clusters that are closest to each other, as the similarity value is showing. The new algorithm and some experimental results are presented in what follows.

Agglomerative algorithm for hierarchical clustering and repetitive similarity values

Input

The set $X = \{w_1, w_2, \ldots, w_n\}$ of n words to be clusterised,
the similarity function $sim : X \times X \to R$.

Output

The set of hierarchical clusters
$C = \{C_1^0, C_2^0, \ldots, C_{n_k}^k\}$

```
BEGIN
    FOR $i := 1$ TO $n$ DO
        $C_i^0 := w_i$
    ENDFOR
    $step := 0$
    $C^0 := \{C_1^0, C_2^0, \ldots, C_n^0\}$
    $C := \{C^0\}$
    WHILE $|C| > 1$ DO
        $step := step + 1$
        $C^{<step>} := C^{<step>-1}$
        $smax := max_{(C_u^{<step>}, C_v^{<step>})} sim(C_u^{<step>}, C_v^{<step>})$
        FOR each $(C_u^{<step>}, C_v^{<step>}) \in C \times C$ , $u <> v$
            IF $smax := sim(C_u^{<step>}, C_v^{<step>})$
                $C_*^{<step>} := C_u^{<step>} \cup C_v^{<step>}$
                $C^{<step>} := C^{<step>} \setminus \{C_u^{<step>}, C_v^{<step>}\} \cup C_*^{<step>}$
            END IF
        END FOR
        $C := C \cup C^{<step>}$
    ENDWHILE
END
```

Taken as input the similarity from table **Table 2** and **Table 3**, with higher rate repetitive value, the results are shown in **Fig 3**.

4. Algorithm to create a similarity tree with respect to hierarchical clusters

Lexical semantics relations play an essential role in lexical semantics and interfere in many levels in natural language comprehension and production. They are also a central element in the organization of lexical semantics knowledge bases.

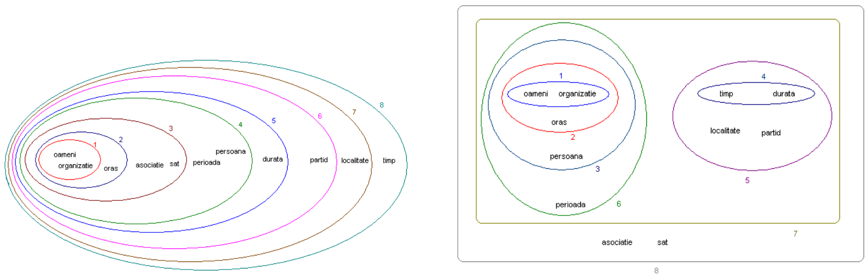


FIGURE 3. Results of agglomerative algorithm for hierarchical clustering on repetitive similarities on experimental data set (table 2 and 3)

Two words W1 and W2 denoting respectively sets of entities E1 and E2, are in one of the following four relations [4]:

identity: E1 := E2,

inclusion: E2 is included into E1,

overlapp: E1 and E2 have a non-empty intersection,
but one is not included into the other,

disjunction: E1 and E2 have no element in common.

These relations support various types of lexical configurations such as the type/subtype relation.

We are interested in constructing a tree structure among similar words so that different senses of a given word can be identified with different subtrees [8]. In what follows we try to model the hierahical clustering algorithm to extract such tree hierarchical structure that we call similarity trees or taxonomy.

For the similarity tree, unification of two clusters in the hierarchical algorithm means to establish a link between two words from the two clusters that are the most similar . The question is now: how to choose those two words when similarity values between words are highly repetitive.

The solution is to find a way to filter the words from a cluster in order to get only one.

The filters we propose are:

- Filter 1: word of maximum similarity
  - choose among candidate words in the two clusters the pairs that have maximum similarity among all pairs of words
- Filter 2: most important words in the cluster
  - choose among candidate words in the two clusters the words that have the sum of the similarities with the other words in the cluster maximum
- Filter 3: most important words for the new cluster

  – choose among candidate words in the two clusters the words that have the sum of the similarities with all the other words in the two clusters maximum
- Filter 4: most important words for the entire set
  – choose among candidate words the words that have the sum of the similarities with all the other words in the entire set maximum

If all those cannot identify a singular word, this indicates that similarity value sets have too many repetitive values that cannot make a distinction among words in some groups. Filtering can be repeatedly applied by using other similarity values sets if it does not obtain an unique word.

Filter algorithm

```
Input
    $CW1 = \{cw_{11}, cw_{12}, \ldots\}$ the set of words to be filtered
    $CW2 = \{cw_{21}, cw_{22}, \ldots\}$ a set of words distinct to CW1
    $W$ : a set of words so that CW1 and CW2 are part of it
         (the set of all considered words)
    $sim : W \times W \rightarrow R$ the similarity function
Output
    $CW1 = \{cw', cw'', \ldots\}$ : the filtered CW1

BEGIN
    IF $|CW1| > 1$ /*** filter 1 ***/
        $msim1 := max\{sim(c1, c2) \mid c1 \in CW1, c2 \in CW2\}$
        $CW1 := \{c1 \mid \exists c2 \in CW2 \ so \ that \ msim1 = sim(c1, c2)\}$
    ENDIF
    IF $|CW1| > 1$ /*** filter 2 ***/
        $msim2 := max\{\sum_{cw2} sim(cw1, cw2) \mid$
                $cw1 \in CW1, cw2 \in CW1, cw1 <> cw2\}$
        $CW1 := \{cw1 \mid msim2 = \sum_{cw2} sim(cw1, cw2),$
                $cw1 \in CW1, \ cw2 \in CW1, cw1 <> cw2\}$
    ENDIF
    IF $|CW1| > 1$ /*** filter 3***/
        $msim3 := max\{\sum_{cw2} sim(cw1, cw2) \mid$
                $cw1 \in CW1, cw2 \in (CW1 \cup CW2, cw1 <> cw2\}$
        $CW1 := \{cw1 \mid msim3 = \sum_{cw2} sim(cw1, cw2),$
                $cw1 \in CW1, \ cw2 \in (CW1 \cup CW2), cw1 <> cw2\}$
    ENDIF
    IF $|CW1| > 1$ /*** filter 4 ***/
        $msim4 := max\{\sum_{cw2} sim(cw1, cw2) \mid$
                $cw1 \in CW1, cw2 \in W, cw1 <> cw2\}$
        $CW1 := \{cw1 \mid msim4 = \sum_{cw2} sim(cw1, cw2),$
                $cw1 \in CW1, \ cw2 \in W, cw1 <> cw2\}$
    ENDIF
```

END

Agglomerative algorithm for similarity tree

```
Input
    The set $W = \{w_1, w_2, \ldots, w_n\}$ of n words to be clustered,
    $S1 : W \times W \to R$ main similarity function
    $S2, \ldots, Sk : W \times W \to R$ other similarity functions
Output
    $T$ similarity tree that respects clusters created by using
    agglomerative hierarchical clustering algorithm

    BEGIN
        $T := \{\}$
        FOR $i := 1$ TO $n$ DO
            $C_i := \{w_i\}$
        ENDFOR
        $C := \{C_1, C_2, , C_n\}$
        WHILE $|C| > 1$ DO
            $smax := max_{(Cu,Cv)} sim(Cu, Cv), u <> v$
            FOR $each\ (Cu, Cv) \in C \times C,\ sim(Cu, Cv)) = smax\ and\ u <> v$
                FILTER($Cu$, $Cv$ , $W$, $S1$)
                FILTER($Cv$, $Cu$ , $W$, $S1$)
                $i := 1$
                WHILE ($i < k$) AND ($|Cu| > 1\ OR\ |Cv| > 1$)
                    $C'u := Cu$
                    IF $|Cu| > 1$
                        FILTER($Cu$, $Cv$ , $W$, $Si$)
                    ENDIF
                    IF $|Cv| > 1$
                        FILTER($Cv$, $C'u$ , $W$, $Si$)
                    ENDIF
                    $i := i + 1$
                ENDWHILE
                IF $|Cu| > 1\ OR\ |Cv| > 1$
                    MESSAGE: $"Undecidable"$
                    END ALGORITHM
                ENDIF
                /* Consider that $Cu = \{cw1'\}$ and $Cv = \{cw2'\}$ */
                $T := T \cup (cw1', cw2')$
                $C := (C \setminus \{Cu, Cv\}) \cup \{Cu \cup Cv\}$
            ENDFOR
         ENDWHILE
    END
```

The algorithm has the advantage of combining the clustering methods with the filetring algorithm in order to obtain similarity trees.

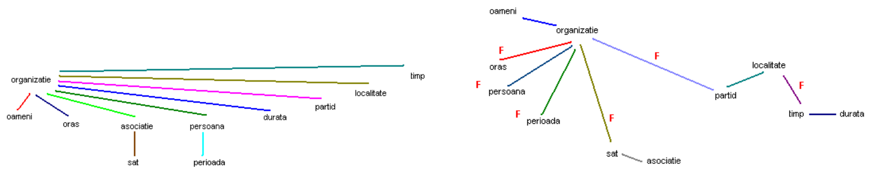


Figure 4. Result of agglomerative algorithm for similarity tree on experimental data set in Table 2 and 3 (hierarchical **AlgOrd**)

Let us construct similarity tree starting with the same similarity values set as used for hierarchical clusters. For those similarity values, the taxonomy algorithm needs supplementary similarity values. Taken as supplementary similarities those from nonhierarchical AlgOrd algorithm, the algorithm is decidable and the two similarity trees that are built for the hierarchical clusters presented above, looks like in **Fig 4**. The big "F" symbol in the figures indicates links that were not decidable without filtering.

5. Conclusions and future research

This paper gives two algorithms to determine hierarchical clusters and similarity trees, starting from untagged corpus data.

We intend to use the method of extracting similarity trees from untagged corpus for semiautomatic building of a IS-A hierarchy for Romanian languaage.

Appendix

	asociatie	durata	localitate	oameni	oras	organizatie	partid	perioada	persoana	sat	timp
asociatie	1	0.96707415	0.95188788	0.98411205	0.98411205	0.98411205	0.95686704	0.97812600	0.97812600	0.99181731	0.94460959
durata	0.96707415	1	0.95188788	0.96707415	0.96707415	0.96707415	0.95686704	0.96707415	0.96707415	0.96707415	0.94460959
localitate	0.95188788	0.95188788	1	0.95188788	0.95188788	0.95188788	0.95188788	0.95188788	0.95188788	0.95188788	0.94460959
oameni	0.98411205	0.96707415	0.95188788	1	0.99846577	0.99893616	0.95686704	0.97812600	0.97812600	0.98411205	0.94460959
oras	0.98411205	0.96707415	0.95188788	0.99846577	1	0.99846577	0.95686704	0.97812600	0.97812600	0.98411205	0.94460959
organizatie	0.98411205	0.96707415	0.95188788	0.99893616	0.99846577	1	0.95686704	0.97812600	0.97812600	0.98411205	0.94460959
partid	0.95686704	0.95686704	0.95188788	0.95686704	0.95686704	0.95686704	1	0.95686704	0.95686704	0.95686704	0.94460959
perioada	0.97812600	0.96707415	0.95188788	0.97812600	0.97812600	0.97812600	0.95686704	1	0.99615956	0.97812600	0.94460959
persoana	0.97812600	0.96707415	0.95188788	0.97812600	0.97812600	0.97812600	0.95686704	0.99615956	1	0.97812600	0.94460959
sat	0.99181731	0.96707415	0.95188788	0.98411205	0.98411205	0.98411205	0.95686704	0.97812600	0.97812600	1	0.94460959
timp	0.94460959	0.94460959	0.94460959	0.94460959	0.94460959	0.94460959	0.94460959	0.94460959	0.94460959	0.94460959	1

TABLE 2. Similarity data set obtained for hierarchical **AlgOrd** algorithm

	asociatie	durata	localitate	oameni	oras	organizatie	partid	perioada	persoana	sat	timp
asociatie	1	0.00000849	0.00000849	0.00000849	0.00000849	0.00000849	0.00000849	0.00000849	0.00000849	0.00003211	0.00000849
durata	0.00000849	1	0.00025204	0.00002015	0.00002015	0.00002015	0.00025204	0.00002015	0.00002015	0.00000849	0.00060790
localitate	0.00000849	0.00025204	1	0.00002015	0.00002015	0.00002015	0.00033500	0.00002015	0.00002015	0.00000849	0.00025204
oameni	0.00000849	0.00002015	0.00002015	1	0.00190216	0.00364963	0.00002015	0.00009627	0.00022050	0.00000849	0.00002015
oras	0.00000849	0.00002015	0.00002015	0.00190216	1	0.00190216	0.00002015	0.00009627	0.00022050	0.00000849	0.00002015
organizatie	0.00000849	0.00002015	0.00002015	0.00364963	0.00190216	1	0.00002015	0.00009627	0.00022050	0.00000849	0.00002015
partid	0.00000849	0.00025204	0.00033500	0.00002015	0.00002015	0.00002015	1	0.00002015	0.00002015	0.00000849	0.00025204
perioada	0.00000849	0.00002015	0.00002015	0.00009627	0.00009627	0.00009627	0.00002015	1	0.00009627	0.00000849	0.00002015
persoana	0.00000849	0.00002015	0.00002015	0.00022050	0.00022050	0.00022050	0.00002015	0.00009627	1	0.00000849	0.00002015
sat	0.00003211	0.00000849	0.00000849	0.00000849	0.00000849	0.00000849	0.00000849	0.00000849	0.00000849	1	0.00000849
timp	0.00000849	0.00060790	0.00025204	0.00002015	0.00002015	0.00002015	0.00025204	0.00002015	0.00002015	0.00000849	1

TABLE 3. Similarity data set obtained for hierarchical **AlgUnord** algorithm

References

[1] S. A. Caraballo, *Automatic construction of hypernym-labeled noun hierarchy from text*, Proceedings of ACL, 1999.

[2] D. Avram Lupşa, G. Şerban, D. Tătar, *From noun's clustering to taxonomies on a untagged corpus*, MPS-Mathematical Preprint Server: Applied Mathematics, 0309004, 2003.

[3] I. Dagan, L. Lee, F. C. N. Pereira, *Similarity-based models of Word Cooccurences Probabilities*, MLJ 34(1-3), 1999.

[4] EAGLES Lexicon Interest Group, A. Sanfilippo, comp., *EAGLES LE3-4244, Preliminary Recommendations on Lexical Semantic Encoding, Final Report*, 1999.

[5] S. Gauch, J. Wang, S. M. Rachakonda, *A corpus analysis approach for automatic query expansion and its extension to multiple databases*, CIKM'97, Conference on Information and Knowledge management, 1997.

[6] C. Manning, H. Schutze, *Foundation of statistical natural language processing*, MIT, 1999.

[7] J. Karlgren, M. Sahlgren, *From words to understanding*, CSLI 2001, pp 294-308, 2001.

[8] D. Lin, *Automatic retrieval and clustering of similar words*, COLING-ACL'98, Montreal, 1998.

[9] C. Oraşan, D. Tătar, G. Şerban, D. Avram, A. Oneţ, *How to build a QA system in your back-garden: application to Romanian*, EACL 2003, Budapest, Hungary, 2003.

[10] V. Pekar, S. Staab, *Word classification based on combined measures of distributional and semantic similarity*, EACL 2003, Budapest, Hungary, 2003.

[11] P. Resnik, *Semantic Similarity in a Taxonomy: An information-Based Measure and its Application to Problems of Ambiguity in Natural language*, Journal of AI Research, 1998.

[12] M. Sahlgren, *Vector-Based Semantic Analysis: Representing Word Meanings Based on Random Labels*, Proceedings of the ESSLLI 2001 Workshop on Semantic Knowledge Acquisition and Categorisation, Helsinki, Finland, 2001.

[13] D. Widdows, *A mathematical model for context and word meaning*, Fourth International Conference on Modeling and using context, Stanford, California, 2003.

Babeş-Bolyai University, Faculty of Mathematics and Computer Science, Department of Computer Science, Cluj-Napoca, Romania

e-mail adresses: davram@cs.ubbcluj.ro, gabis@cs.ubbcluj.ro, dtatar@cs.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLVIII**, Number 2, 2003

SPATIAL VIEW OF 3D OBJECTS USING STEREOGRAMS

V. PREJMEREAN, S. MOTOGNA, AND V. CIOBAN

ABSTRACT. We present a method to view in space a three-dimensional solid body using polygonal representation. This method has been chosen because it allows a visualization of a body without other instruments. We prefer the screen visualization instead of paper visualization, because we can also produce an animation, for example by rotating the body (arround the axis, using the keyboard), increasing the image reality.

1. Introduction

There are several methods to visualize a body in space, in the way we see it in reality. Stereograms allow such a view without other special instruments, just a certain skill to view these objects.

A similar method, that can be easy applied, needs for visualization a pair of glasses with different colored lens. The spatial view is based on the fact that each eye sees the same body, but from a different angle, rotated. It's obvious that we have to draw two images simultaneously (one for each eye) and then to distinguish them (each eye should see its corresponding image). This technique can be found in some spatial geometry books, that also contain the glasses, and the effect is quite spectacular. The stereograms have a totaly different functioning principle (we shall see this difference in their construction), an object in space, as in reality, having the third dimension (the depth). Unfortunately that bodies are not moving! Working on the screen, this thing will become possible, and even more, the user can obtain the desired move.

Either on paper or on screen the three-dimensional objects are drawn after they have been transformed through a projection in a real (bidimensional) plane, then a new transformation is applied through which the plane image is framed in a window of the paper or of the screen. The screen image is no longer seen in space (it is a plane image), we just imagine how the initial three-dimensional object looks [5]. Oftenly the same plane image can lead to false interpretations (optical

Received by the editors: October 22, 2003.

2000 *Mathematics Subject Classification.* 68U05, 65D20.

1998 *CR Categories and Descriptors.* I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling – *Curve, surface, solid, and object representations*; I.3.7. [**Computer Graphics**]: Three-Dimensional Graphics and Realism – *Animation.*

illusions) since the reverse transformation is not unique. In the following we will refer to the screen representations, because these objects can be rotated arround the coordinate axis using the keyboard, increasing the reality of the image. We will argue why we have chosen the perspective projection instead of the parallel one, besides the fact that is the real one. If we do not consider the animation, the images (stereograms) can be printed and viewed in space, but statically.

2. The principle of Stereograms

Visualizing a stereogram doesn't need any extra instrument, only a certain skill. Practically, you need to look behind the screen (at a certain distance), each eye seeing two pyramids, as in figure 1. When two of the four pyramids will overlap (only three will be seen), the one from the middle will be seen in space.

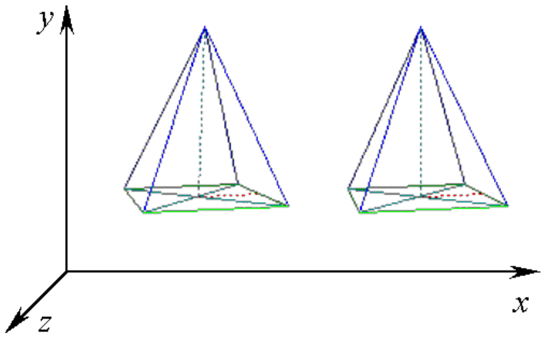


Figure 1. An example of a stereogram

The two drawn objects will not be identical! If this would be the case, then the middle object will be better seen, but not in space. This image will be plane too, as a picture, in which we just imagine its depth, but we don't see it in space, as in reality. In order to have a spatial view, the two images must be different, similar to reality when each eye sees the same image but from a different angle, allowing to sense the third dimension, the depth. If the second object (the right-hand side pyramid) would have been obtained rotating the first object arround the Oy axis with some angle (assuming that the observation point is on Oz coordinate, that expresses the depth), then the resulting object is not a stereogram. If we would be able to see each object with the corresponding eye, then we will see the spatial representation of the object. An easy method to achieve this purpose uses glasses with different colour lenses, allowing to filter the image. The two objects will have different colours corresponding to the lenses' colours. This method has at least two disadvantages: we need the special glasses, and we have only some types of viewable images (depending on colours). Even for this method, a certain skill in visualizing the object is needed, different from person to person.

Our method overcomes the two disadvantages, being able to use all combinations of colours in the image, just having some training to visualize the stereograms.

The principle of stereogram visualization is: the shorter the distance p between two pixels is, the closer the point seen in space (in depth) is to the screen (distance d), as shown in Figure 2. If we want this point to be farther to the observer (and implicitely to the screen), then we must increase the distance between the two points [1,2,3]. From Figure 2, we can easily deduce that:

$$d(p) = e \times p/(g - p)$$

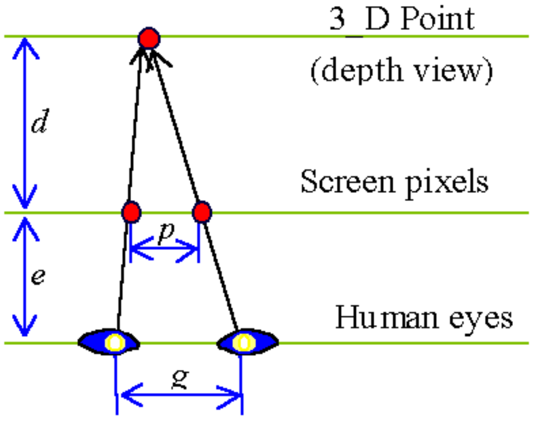


FIGURE 2. Visualizing a stereogram

From a practical point of view in constructing a stereogram, we are more interested in the inverse function to determine where to put a pair of pixels on the screen: which distance p do we need to obtain a desired depth d. The inverse equation is:

$$p(d) = g \times d/(d + e)$$

If we consider the identical pair figure, at a distance p_0, then the composed figure will be in a plane situated at the distance $d_0 = d(p_0) = e \times p_0/(g - p_0)$ from the screen. But what we want is to have some points nearer, and some farther. If we want to have a point in a farther plane, at the distance $d = d_0 + \Delta d$, we must know how much we should increase the distance p_0. So, we have to compute Δp

$$\begin{aligned}\Delta p &= p(d) - p(d_0) = p(d_0 + \Delta d) - p_0 = g \times (d_0 + \Delta d)/(d_0 + \Delta d + e) - p_0 = \\ &= g \times (e \times p_0/(g - p_0) + \Delta d)/(e \times p_0/(g - p_0) + \Delta d + e) - p_0\end{aligned}$$

and obtain it as an expression of pixels:

$$\begin{aligned}&\Delta p_p = [\Delta p \times Nph/Dsh],\ \textit{where} \\ &\qquad Nph = \textit{the number of horizontal pixels} \\ &\qquad Dsh = \textit{horizontal size of the screen}\end{aligned}$$

The above equations also depend on the distances e and g:

- the distance e is in the range 50 to 70 cm (≈ 20"-25");
- the distance g representing the stereographic ability of a pupil gauge, with values around 6.5cm (2.56").

It is necessary to note that the distance between to pixels from a pair is strictly positive, and obviously could not exceed the pupil gauge: $(0 < p = p_0 + \Delta p < g)$. For example, if we want a distance of approximatively 5cm (≈ 2"), considering the distance between two pixels of 0.042cm (0.0166"), then the initial value of p_0 should be 120 pixels. If the stereographic ability of the user is good, then this distance can be increased, without exceeding g, and if this stereographic quality is low, then the decrease of the initial distance p_0 is recommended, but not too much in order to avoid the overlaping of the two images (in which case the stereogram visualization is difficult).

3. The construction of a stereogram

Firstly, we will chose the screen windows according to the conditions mentioned in the previous section. In the first window we will represent the first object in a perspective projection, and in the second window the same object will be projected modifiying the x coordinates corresponding to the depth view, as described above.

In figure 3, we considered an example in which we have increased the distance AA' and decreased the distance BB', yielding to a move of the point A in a farther plane and of point B in a nearer plane $(AA' > VV' > BB')$. The resulting effect is a triangle visible in space and no longer parallel with the screen.

One may notice that by decreasing the base AB we obtain a rotation of the triangle in one direction, and by increasing the base we obtain the rotation in the opposite direction, as shown in Figure 4.

We also mention that as much as we further the extremity of a segment from the screen, the longer the segment becomes (in projection), as in the case of the segment VB from the left image in Figure 4. This seems like an anomaly, because in reality the farther the object is, the smaller it looks. In order to overcome this effect we recommend the perspective projection by which the farther sides will be

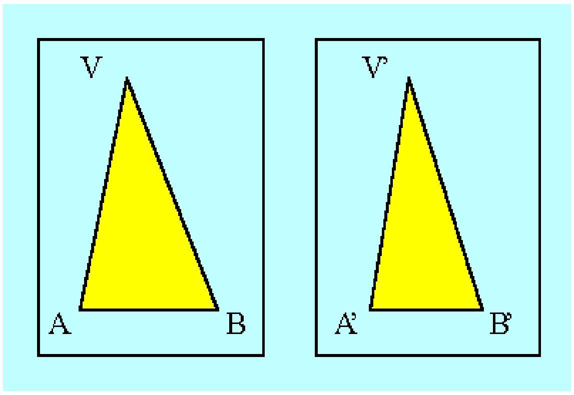


FIGURE 3. Construction of a surface

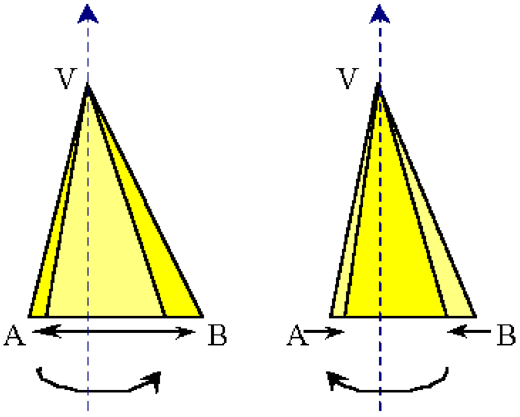


FIGURE 4. Rotation of a surface

drawn smaller, and when increasing their lenghts they do not exceed the lengths of the nearer segments. Therefor, it is obvious why the perspective projection is preferred for spatial view.

Practically, for the first figure from the stereogram we follow the standard steps of representing a polyhedron:

(1) establish the screen window $V_1(u_1, v_1, u_2, v_2)$; for example
$Viewport(200, 100, 300, 200)$;
(2) determine the real window (of the projected figure), namely the minimal rectangle containing all the projections of the points;

(3) represent each segment (side) with its correponding characteristics: line type, colour,...

The second figure is drawn in a similar way, with the following differences:

(1) establish the screen window $V_2(u_1+p_0, v_1, u_2+p_0, v_2)$, such that:
   - $u_2 < u_1 + p_0$ to avoid windows intersection (and consequently figure overlaping);
   - $p_0 < g$ such that the distances between two pair-pixels is less than the pupil gauge;

   for example: $ViewPort(340, 100, 440, 200)$, so $p_0 = 140$;

(2) the real window computed before is preserved, since the body and the projections characteristics remain unchanged;

(3) represent each side of the body, performing the necessary corrections for the x coordinate of each pixel based on the z coordinate value of the corresponding point (assuming that the observer is on the positive semi-axis Oz). These corections can be made either increasing the horizontal distances between the pixels representing the vertexes without exceeding the distance g, either decreasing these distances, taking care not to overlap the left figure, that will affect the "readability" of the stereogram.

Let $P(x, y, z) \in \Re^3$ be the point we want to represent, and its projection $P'(x', y') \in \Re^2$ will be determined in the following way:

$$x' = x \times (\delta - q)/(\delta - z)$$
$$y' = y \times (\delta - q)/(\delta - z)$$

where δ represents the distance of the observer from the origin $Obs(0, 0, \delta)$, on the positive semi-axis Oz, and q represents the distance of the projection plane from the origin, that is parallel with the plane xOy. These conditions skip any other geometrical ransformations [4,5], and allow future rotations of the body, such that it can be put into a certain desired position. We have also taken into consideration that the observer should be placed at a considerable distance from the figure (greater than the biggest side of the figure) such that the body will not be represented too irregular, but also not too big, when the projection will seem like a parallel one.

The pixel $P"(u, v) \in V_2$ from the declared screen window obtained through window transformation is given by:

$$\begin{aligned} u = &\quad (x' - x_1)/(x_2 - x_1) \times (u_2 - u_1) + u_1 + p_0 \\ v = &\quad (y' - y_1)/(y_2 - y_1) \times (v_2 - v_1) + v_1 \end{aligned}$$

for a screen window $V_2(u_1+p_0, v_1, u_2+p_0, v_2)$ and a real window $W(x_1, y_1, x_2, y_2)$, already determined for the first figure.

In order to have a spatial view we must apply a depth corection, yielding to the point $P_S(u - \Delta p_P, v)$.

Although the computing equations have been presented in section 2 (including for Δp_P) and can be used, we consider that is much simpler to make a direct corection, based on the coordinate z of the point P using the following formula:

$$\Delta p_P = (z - min\{z_i\})/\Delta z \times k$$

where:

- Δz is the height:
  $(\Delta z = max\{z_i | P_i(x_i, y_i, z_i) \in V\} - min\{z_i | P_i(x_i, y_i, z_i) \in V\})^1$
- k is the depth amplification constant, for example 1.5. This constant can be determined either by objective factors (the ratio $\Delta x/\Delta z$), either subjective factors: the distances g and e from Figure 2.

This correction is applied only in one direction because $\Delta p_P \in [0, k]$, so it allows just to see some points nearer to the screen. In order to view them farther, we will modify the equation in the following way, taking care not to exceed the distance e:

$$\Delta p_P = ((z - min\{z_i\})/\Delta z \times 2 - 1) \times k, so\ \Delta p_P \in [-k, k]$$

If the observation will be performed in front of the screen (as we will explain later), then the windows can be choosen as the two halves of screen (left and right), and k can be greater, for example 50. If this constant is too big, for example 100, then certain points are formed to closer to the observer and the figure is more difficult to be seen. This kind of visualization also require changing the sign of the constant, and therefor of the correction.

It is possible to apply a corection such that the distance between the objects could be modified, in order to give a kind of freedom to the application regarding the observation ability of users. The final horizontal coordinate of a pixel will be:

$$u' = u - \Delta p_P - Npo$$

where Npo represents the closing distance, expressed in pixels.

The depth correction can be applied even before the window transformations, in the projection plane, in the following way:

$$x' = x - \Delta p_R, \text{ with}$$
$$\Delta p_R = u^{-1}(((z - min\{z_i\})/\Delta z \times 2 - 1) \times k)$$

where u^{-1} is the inverse transformation, from screen window $V_2(u_1 + p_0, v_1, u_2 + p_0, v_2)$ into the real window $W(x_1, y_1, x_2, y_2)$:

$$x' = (u - u_1 - p_0)/(u_2 - u_1) \times (x_2 - x_1) + u_1.$$

^{1}V represents the set of body vertexes

Of course, in this situation, the depth correction will not be applied in the screen window, so:

$$u' = u - Npo.$$

The effect is more signifying if we draw three pyramids: the initial one, one on the left and one on the right at different distances. Seen with one eye, there will be six images: two of them will overlap, and will be seen in space (in the middle), and two will be simple, in plane, visible on sides. As a consequence, there will be four pyramids, the middle ones in space and not at the same distance, the left one will rotate in the reverse direction and its projection is incorrect drawn (the farther side is seen longer, therefor incorrect). In order to have a valid representation we must modify the equation for the left pyramid with + yielding the point $(P_S(u+\Delta p_P, v))$ obtaining the same direction (see Figure 4) and a correct projection.

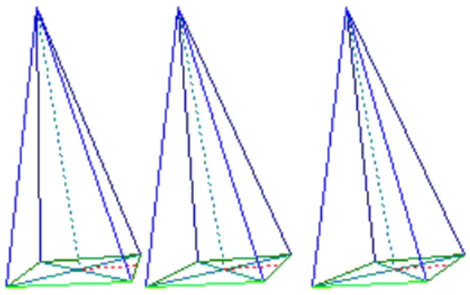


Figure 5. Bodies at different distances

An interesting remark is that if we apply different values of Npo, then the two pyramids (which now will be rotated simultaneously, and in the same direction, and will be correctly visualized through a perspective projection) will no longer be in the same plane (one will be closer to the observer than the other one), and even more will have different dimensions (surprinsingly, the farther pyramid will look bigger, contrary to what be would probably expect), as shown in Figure 5.

An even more interesting possibility is to visualize a stereogram in front of the screen (not in the back, as in our previous examples), as shown in figure 6. This case offers the great advantage of visualizing a stereogram represented on a big surface (for example, projected on a wall), where it is obvious that the distance p is greater than the pupil gauge g.

If we want to visualize the figure in front of the screen, then the projection corrections will be applied inversely to the back screen representation, namely positively for the right figure: $P_S(u + \Delta p_P, v))$ and negatively for the left figure:

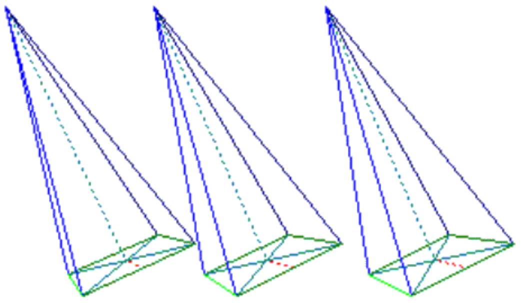


FIGURE 6. Pyramides in front of the screen

$P_S(u - \Delta p_P, v))$. The effect is somehow stronger, since the objects are clearer and nearer.

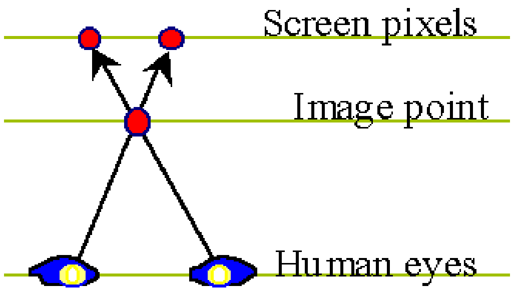


FIGURE 7. Visualization in front of the screen

4. Performing Animation

It is well known that animation helps understanding a spatial image, since the dynamic projection is an important factor in increasing its reality. The fact that the image is rotating, for example the pyramid is rotated arround its height, helps us imaginaing this pyramid in space, even without representing it as a 3-dimensional object, since a nearer point is rotated in one direction and a farther one in rotated in the opposite direction. This effect is increased even more in case of a stereogram, yielding to a better visualization of the image.

In certain cases it is more convenient to allow the user the possibility to manipulate the represented object through rotations, using the keyboard:

- the keys **O** and **P** to rotate the object arround the Oy axis with some angle α (not too big), in one direction, respectively in the other direction;
- the keys **Q** and **A** to rotate the object arround the Ox axis with some angle β in one direction, respectively in the other direction;
- the keys **S** and **D** to rotate the object arround the Oz axis with some angle γ in one direction, respectively in the other direction;

In order to simulate the increase of the rotation speed, that can be performed arround the weight centers, the angles α, β and γ can be automatically increased after each rotation. In this way, holding one key, after we visualized the body in space, this will be rotated correspondingly to the pressed keys faster and faster. The coordinates of the point G, arround which we execute the rotation, can be determined as the arithmetic mean of the body vertexes. The rotation will be executed arround some lines parallel with the axis, that cross through the point G. This rotation can also help us to visualize the stereogram.

In order to have a continous move of the body we use two active and two visual pages for alternative draw and visualization of the images.

As a final remark, we note that in observing a stereogram at the beginning everything seems “foggy” and then it becoms more clearer, as some real objects we could touch. This observation is easier for a trained viewer, so “Good luck!”.

5. Remarks

The article contains figures in black and white, losing some image quality, and of course its animation. That’s why we have put the examples from this article on the web page of the publishing journal: `http://www.cs.ubbcluj.ro/∼studia-i`.

References

[1] A.A. Kisman, *Random Dot Stereograms*, Kinsman Physics P.O. Box 22682, Rochester, N.Y., 1992.

[2] V. Prejmerean, V. Cioban, *3D Images Simulation through Stereograms*, Research Seminars, Seminar on Computer Science, Preprint 2/1996, pp. 75-78.

[3] V. Prejmerean, *Modelling Stereograms by Matrix Grammars*, Research Seminars, Seminar on Computer Science, Preprint 2/1997, pp. 173-178.

[4] V. Prejmerean, *Computer Graphics and Image Processing*(in, Universitatea de Nord Baia-Mare, Facultatea de Ştiinţe, 2000.

[5] A. Watt, *3D Computer Graphics*, University of Sheffield, Addison Wesley, England, 1993.

Babeş-Bolyai University, Department of Computer Science, RO-400084 Cluj-Napoca, Romania

E-mail address: `per@cs.ubbcluj.ro`

E-mail address: `motogna@cs.ubbcluj.ro`

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLVIII**, Number 2, 2003

ON INDIVIDUAL PROJECTS IN SOFTWARE ENGINEERING EDUCATION

MILITON FRENŢIU, IOAN LAZĂR, AND HORIA F. POP

Abstract. A study of the individual projects is performed and some thoughts on the importance and the development of these projects are given. The projects are written by second year undergraduate students as a requirement in their curriculum, and by graduate students working for their companies, for comparison. The principal components method is used as Data Analysis Technique. Some consequences on the education activity are considered.

Keywords: Student Projects, Education, Software Engineering, software metrics, measurement, data analysis technique. fuzzy clustering

1. Introduction

Undergraduate computing courses having a project component (that attemps to convey some of the aspects of a real-life development project) often concentrate on the final product, rather than the process by which it is achieved [17]. However, it is important that the students should see the necessity of all phases of the life-cycle of a project, they must be accustomed to produce the documents for all these phases. We think it is important for undergraduate students their first project be PROCESS-ORIENTED activity.

There are two projects in the curricula of undergraduates in Computer Science at Babes-Bolyai University. First is an individual project, and a second is a Team project in the third year. Both projects are closed projects [13]. Also, it is supposed that each student will conceive an application connected to the diploma work at the end of studies (fourth year). The aim of individual projects at the Babes-Bolyai University is to give the students the opportunity to accomplish a simple project fulfilling all the steps of the life-cycle [5]. It is planned in the third semester, after the students have learned Algorithms and Pascal programming in the first

Received by the editors: December 10, 2003.

2000 *Mathematics Subject Classification.* 68N30.

1998 *CR Categories and Descriptors.* D.2.3 [**Software**] : Software Engineering – *Coding Tools and Techniques*; I.5.1 [**Computing Methodologies**] : Pattern Recognition – *Models – Fuzzy set*; G.3 [**Mathematics of Computing**] : Probability and Statistics – *Data analysis*.

semester, and Object-oriented programming (in C++), and Data Structures in the second semester.

The subject runs in the first semester of the second year, and is the first project done by an undergraduate student. It is a small project, of about a few hundreds up to 3000 statements. The main purpose is to make the students accustomed to finalize a project and to write all needed documents: requirements, specification, design, code, testing, and user manual, and, very important, to obey the deadlines. Summarising, the main objectives of the individual projects are:

- going through all steps of the life-cycle;
- observing the deadlines;
- writing a clear documentation for all steps;
- verifying and validating the program; inspections and reviews are considered very important means of eliminating errors;
- showing to the students when their programs are insufficiently tested.

The project extends over a fourteen week semester and in each week there is one hour practicum session in a computer laboratory. Students are also expected to complete an extra work at home to meet the requirements of the subject.

At every deadline the student have to deliver the corresponding document [5]. Missed deadlines are penalized; it is considered a decrease of the grade for the corresponding activity by one point for every late week.

Individual project features (document outlines) and the included aspects (each to be completed by students in two weeks) are:

Requirements specification: a conceptual model, functional requirements, user interface requirements, error handling;

Design specification: system architecture, and detailed module specification;

Test plan: a system level testing plan, a detailed unit test plan, and test results;

Coding: correct translation of the design and commented source code;

User manual: an introduction, general principles for using the software, tutorial on how to use the software, a list of software's functions and simple installation instructions;

Project evaluation: comments on the quality of the actual product (what was done well and what could be improved in future projects).

A complete documentation for each of the above mentioned phase is required to support the undergo software engineering process.We require a clear and complete documentation from the students, and we consider this as one of the main gain acquired by students through this project. Their attention has to be turned from immediate coding to correct approach of all life-cycle phases, with an accent on complete and accurate specification and clear and correct design. Is well known that the most difficult errors, and their greatest part, as well, are not due to bad coding, but to incorrect design. We must convince the students by all means that a serious programmer is not the one that starts coding immediately as he gets the requirements. Good programming means not jumping any step of the life cycle, a good design is crucial for the next project, and the documents are needed for the entire activity as early as possible.

An important activity required to the students was the inspection of the documents written by their colleagues. It is known [6, 10, 14] that many shortcomings of the software process may be eliminated by inspecting all the steps of the life-cycle, starting with the requirements. This activity persuades the students on the necessity and importance of documments for all phases, and gets them accustomed to analyse other people's documents. Therefore, forcing the students to inspect the documents of other people is important not only for software engineering profession, but also for education. By inspection, students have the opportunity to see how others think and write programs, and to discover some of their errors. Also, the teacher has the possibility to discuss with the students about errors, shortcomings and difficulties, to improve the software process by various discussions and questions about them. The teacher has the possibility to provide to students a better way of solving problems, to force them to elliminate defects, and to improve their ways of programming.

Some important 'real-life' project features [12] are: (a) working with real users, (b) developing a working prototype, (c) completing a running system and (d) writing a formal verification and validation report. This individual project does not address the issues of 'real-life' projects, but it is mainly educationally oriented. The students do participate, during their third year of academic studies, to a group project which is 'real-life' oriented.

2. Observed attributes and Data collection

The study is based on 28 finished programs produced by second year undergraduate students as part of their requirements curriculum, and other 8 real products produced by graduates students at their software companies. The observed attributes were estimated by master students attending the course "Software Metrics". Certainly the measures associated to the attributes are the subjective evaluations of these master students about the corresponding attributes. We may accept that the postgraduate students are not experienced programmers, but they have finished a similar project three years earlier, and other two projects in their third and fourth year. Many of them are also working for software companies. Moreover, their evaluation was inspected by the first author and a few corrections were made (where obviously erroneous evaluations have been noticed).

The postgraduate students form a Master group in "Component-Based Programming" that studies the subject "Software Metrics". The definitions of the above considered attributes were given there, and they are inspired by, and can be found in the literature [2, 9, 14]. As an exercise they had to evaluate one project as a requirement for their seminaries at "Software Metrics", and theirevaluation was discussed with the first author of this paper. The analysed projects of the undergraduate students were seen beforehand and graded by some other teaching assistants and their grades are not influenced by this study.

Therefore, students involvement is twofold. On one side the undergraduate students of the second year learned to specify, design, code and test a complete program, on the other side, the master students learned to analyse software documents, to measure software attributes. Data collection is an important software measurement activity, and, since it is not easy to obtain access to real projects, this was another possibility to get used to evaluate software attributes. And we remarked it was a very useful activity, since the master students were very critical about the analysed projects, about the clarity of the documents and of the design, about the absence of the comments in the code. And they remembered they had the same shortcommings three years earlier.

The projects were analysed observing the attributes given in Table 1.

A1	requirements description	A18	number of classes
A2	good specification	A19	number of methods for all classes
A3	function points	A20	changeability (modifiability)
A4	clarity of design	A21	structuredness
A5	correctness of design	A22	testability
A6	completness of design	A23	reliability
A7	diagrams of design	A24	efficiency
A8	modules specification	A25	extensibility
A9	algorithms description	A26	adaptability
A10	lines of code	A27	clarity of documentation
A11	no. of comments	A28	maintainability
A12	good use of comments	A29	simplicity
A13	good use of free lines	A30	usability
A14	indentation	A31	portability
A15	good names	A32	quality
A16	readability	A33	average of weighted methods per class
A17	comprehensibility	A34	depth of inheritance

TABLE 1. The attributes observed for the software projects analysis

The attributes A10 and A11 were automatically measured by computer. All the others were estimated by master students attending the course "Software Metrics".

All metrics have the values in the interval [0, 10], where 0 is for ”very bad” (or not present at all), and 10 for “excellent”.

Certainly, these grades are subjective estimates on the projects. Nevertheless, we consider that the dependence between attributes is preserved in these data, and the strong dependence between almost all attributes and the knowledge of the authors (reflected in A34) is preserved. We may use these data and the results to draw some useful conclusions on the organisation of the software development process and the way it may be improved.

The attribute A12 refers to the documentation done by comments. It takes into account if the specification of each module is reflected through comments, if the meaning of each variable and object is explained by comments, if the invariants and other important explanations are given by comments.

3. Data Processing

We have analyzed a data set composed of 36 projects, characterized by 34 software metrics attributes. The 36 projects consist of 28 educational undergraduate projects and 8 real projects (namely, 12, 13, 21, 25, 28–31). In order to save the editorial space, the whole data set, as well as complete computational results, are not published here, but are available from the authors[1]

We have run a few experiments in order to detect the proper relations among the data. We were interested in studying the fuzzy cluster substructure of the data set, as well as the fuzzy cluster substructure of the set of attributes [16]. A special note with respect to attributes 18, 19, 33 and 34. These are characteristic to object-oriented approached projects, and are not relevant for other projects. As such, we have considered our study in two scenarios: on one side, we have considered the value of these attributes to be zero for the projects without object-orientedness. On the other side, we have marked the values of these attributes as missing.

In the first case we have used the Fuzzy Divisive Hierarchic Algorithm, with the Euclidean metric [15], and in the second case we have used the optimal completion strategy as outlined in [7], but in the same framework of the Fuzzy Divisive Hierarchic Algorithm. The results in the two cases are practically identical, allowing us to consider as reasonable fact to assign a value of zero to an object-oriented attribute in the case of a non-object-oriented project.

The final defuzzyfied hierarchy corresponding to the optimal fuzzy cluster substructure of the set of projects is described in Table 2. We remark grouping of all student projects in subclasses of the class 1.1.1, and the separation of the industrial

[1]The primary data is not given here, but can be seen, together with full data analysis results, at the web address `http://www.cs.ubbcluj.eo/ mfrentiu/articole/project3.html`.

projects in the other classes, along patterns of similarity to the student projects. Thus, projects 12, 13, 21, 25 have been grouped as class 1.1.2, as the most similar projects to the student group; project 31 forms class 1.2.1, and projects 28 and 30 form class 1.2.2, both classes considered more distant to the student group. Finaly, project 29 forms class 2., showing a clear separation of the whole set of projects. On the other side, the student projects have been further divided, the most notable split being among projects 3, 5, 7, 8, 15, 32–34, 36 (class 1.1.1.) and 1, 2, 4, 6, 9–11, 14, 16-20, 22-24, 26, 27, 35 (class 1.2).

Class	**Members**
1.1.1.1.1.1.	5 8
1.1.1.1.1.2.	3 15 33
1.1.1.1.2.1.1.	7
1.1.1.1.2.1.2.1.	34
1.1.1.1.2.1.2.2.	36
1.1.1.1.2.2.	32
1.1.1.2.1.1.	1 17 20
1.1.1.2.1.2.1.1.1.	26
1.1.1.2.1.2.1.1.2.	35
1.1.1.2.1.2.1.2.	6 27
1.1.1.2.1.2.2.	9 16 22 23
1.1.1.2.2.1.1.1.	18
1.1.1.2.2.1.1.2.	2
1.1.1.2.2.1.2.1.	11
1.1.1.2.2.1.2.2.	14 19
1.1.1.2.2.2.1.1.	24
1.1.1.2.2.2.1.2.	4
1.1.1.2.2.2.2.	10
1.1.2.1.1.	13
1.1.2.1.2.	25
1.1.2.2.1.	12
1.1.2.2.2.	21
1.2.1.	31
1.2.2.1.	30
1.2.2.2.	28
2.	29

TABLE 2. Final defuzzyfied partition corresponding to the optimal fuzzy cluster substructure of the set of projects

The final defuzzyfied hierarchy corresponding to the optimal fuzzy cluster substructure of the set of attributes is given in Table 3(a). The attributes given in paranthesis are not clear-cut members of those classes, but have relevant fuzzy membership degrees so that they should be taken into account, as well.

We first remark that quite a large number of attributes have not been split. The attributes 1, 2, 4–9, 13–17, 20–32 (class 1.1.1.1.1) may actually correspond to a single inherent property of programming projects, property that is expressed through more different attributes. Other than these attributes, attribute 12 has membership degree 0.42 (as compared to 0.56, the membership degree to its class (!)), and attribute 33 has membership degrees of 0.27 (as compared to 0.30, the membership degree to its class (!)).

Other than this large class, we identify a few relevant groups: the most significant separation is of attribute 10 (class 2), the program size (lines of code). At the next level, we identify attribute 11 (class 1.2.2), associated to the program size as well (lines of comments), and attribute 19 (class 1.2.1), representing the total number of methods. A supporting member of class 1.2.1 is attribute 3, with quite a relevant membership degree of 0.30 (as compared to 0.70, the membership degree to its class).

On the other side of the tree, we identify the class 1.1.1.1.2 (formed by attributes 12 – comments accuracy, 33 – weighted methods per class, and 34 – depth of inheritance; attribute 7 is supporting member, with a membership degree of 0.30, as compared to the membership degree of 0.69 to its class); class 1.1.1.2 (attribute 18 – number of classes); class 1.1.2 (attribute 3 – function points; attribute 19 is a supporting member, with a membership degree of 0.33, as compared to the membership degree of 0.67 to its class).

In order to verify the relevance of the attributes in the presence or absence of industrial projects, we have removed the industrial projects from our attributes classification. The results are given in Table 3(b). As in the precedent case, the attributes given in paranthesis are not clear-cut members of those classes, but have relevant fuzzy membership degrees so that they should be taken into account, as well.

We remark an extremely similar hierarchical clustering structure. If we consider the shared attributes (i.e. those with mixed mebership degrees), the similarity is even higher. Thus, the main group of attributes, 1, 2, 4–9, 12–17, 20–32, formerly the class 1.1.1.1.1, forms now the class 1.1.1.1.

As well, the attributes 18, 19, 33, 34, formerly roughly with classes 1.1.1.1.2 and 1.1.1.2, form now the class 1.1.1.2. This class has the attribute 12 as a supporting member, with a membership degree of 0.30 (as compared to the membership degree to its own class, 0.69).

We may conclude that industrial projects have a marginal, even insignificant influence on the classification of the project attributes, confirming the overall influence of knowledge level to the way people approach the software process.

On the other side, by taking into account the supporting members of fuzzy classes, we may finally consider five main groups of attributes:

- 10 (lines of code);
- 11 (lines of comments);
- 3, 19 (program complexity);
- 18, 33, 34 (object-oriented attributes);
- 1, 2, 4-9, 12-17, 20-32

Even if only few projects out of a total of 36 are object-oriented, three of the four object-oriented attributes group together (18, 33, 34), and the fourth, the total number of methods for all classes (19), is consistently grouped together with the function points (3), both showing an impact of the overall program complexity.

Class	**Members**
1.1.1.1.1.	1 2 4 5 6 7 8 9 13 14 15 16 17 20 21 22 23 24 25 26 27 28 29 30 31 32 (12, 33)
1.1.1.1.2.	12 33 34 (7)
1.1.1.2.	18
1.1.2.	3 (19)
1.2.1.	19 (3)
1.2.2.	11
2.	10

(a)

Class	**Members**
1.1.1.1.	1 2 4 5 6 7 8 9 12 13 14 15 16 17 20 21 22 23 24 25 26 27 28 29 30 31 32 (12)
1.1.1.2.	18 19 33 34 (12)
1.1.2.	3 (19)
1.2.	11
2.	10

(b)

TABLE 3. Final defuzzyfied partition corresponding to the optimal fuzzy cluster substructure of the set of attributes: (a) with all the 36 projects considered; (b) only with the undergraduate projects

Figure 1 presents the 2D projection of the set of 36 projects along the first two principal components, as determined using the well-known Principal Components Analysis applied to the correlation matrix. The figure clearly displays three categories of projects: the two isolated industrial projects (25 and 29), a second well-separated group of industrial projects (12, 13, 21, 28, 30, 31) and the homogenous group of student projects. It is important to note that the main group of educational projects presents a clear linear trend, supporting the conclusion that most of the considered attributes correspond to a single factor, identified as the overall level of knowledge of the student.

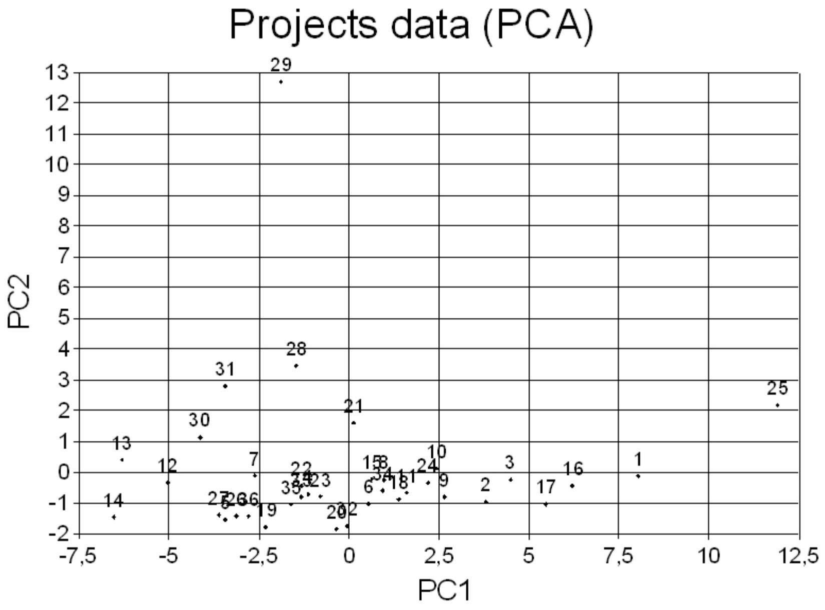


FIGURE 1. Principal Components Analysis of the set of 36 projects. The set is projected along the first two principal directions

4. CONCLUSIONS

In this paper we aimed to comparatively study educational and industrial projects. We acknowledge the fact that there are different factors contributing to these projects. On one hand, the purpose of educational projects is to train students into approaching a problem in an ordered manner. As such, the students will have to use different knowledge gathered at different classes. On the other hand, the purpose of industrial projects is to produce final software products that solve a real problem, required by a real customer, and distributed to the real world. These differences among the purposes of the two groups of projects contribute to their grouping in different fuzzy classes.

As it has been expected, the fuzzy clustering procedure confirms that all the qualitative attributes are dependent on the programmers general knowledge, the main such factor.

It is interesting to remark that, generally, the set of attributes have been split along the lines of object-orientedness. Apparently, the object-oriented software metrics are consistent with the function points attribute; the two size attributes group closely, while all the others are grouped in a single class, allowing us to conclude that the students follow in the same manner all the programming rules and habits they are taught. We conclude that the object-oriented software metrics attributes measure, actually, the complexity of the object-oriented project. It is, thus, needed to concentrate on providing object-oriented attributes that measure the quality of an object-oriented project, as well.

We have analyzed software products made by undergraduate students. We are confident that the results cannot be extrapolated to large software systems. But they can certainly be used towards a better students formation and may be used as effective didactic materials, especially with the "Software Metrics" course. Even if we insist with the first year students on the necessity to develop their own programming style, to obey a few important programming rules [4], the students are skeptical, they are simply happy that their programs "work", they do not like to write comments, or to insist on a good design and Pseudocode algorithms, or documentation.

By analyzing the primary data, we may observe that students do not like writing comments. However, a certain progress is remarked from one generation of students to the subsequent: the necessity of comments appears strongly with this year's students as opposed to the last year's and two years' ago [3].

As well, we remark that undergraduate students refrain from writing complete documentations. Their documents are generally superficial, written on short notice, only to fulfill a requirement lined out by the professor. The students do not generally consider writing project documentations as part of their natural thinking process, required in order to produce effective projects. As well, project documentations are often mixed, in the sense that ideas that are naturally part of the design and implementation process are included in the specification documentation. Nevertheless, at the end of the activity the majority of the students consider that the main gain obtained through this project is understanding the importance of, and learning to write a complete documentation.

Completely different is the approach of real projects programmers (in our study, graduate students) on the necessity of a complete and correct documentation, its usefulness, and the effect of an adequate programming style on final projects.

We remark that the theoretical and practical knowledge of the average student has improved from one generation to the other. This is a confirmation of the success of our stepwise approach towards students education of projects development.

We observe that very few undergraduate students practice object-oriented programming. They have learned it in the second semester of the first year, but are not used to practice it, or they do not feel its necessity or advantages. Why is it so, is a natural question we must try to answer.

We consider useful to provide the students with an effective model, possibly the best project of the previous generation. On the other side, it may be useful to consider as project topics for a part of the students, the requirement to work on improving projects already written by students of the previous generation. This should be a step in the right direction, of improving students knowledge on developing projects.

The continuous improvement of the educational process is the ultimate purpose and goal of any teacher. And aquiring correct habits of developing a programming product is one of the major issues a computer science graduate will have to face. On the other side, a thorough study of the process of projects development, and complete data collection and its accurate interpretation should become a part of our educational activity.

We intend for the next year to improve these projects, asking to the new generation to maintain the existing projects, correcting the possible discovered errors, adding new functions, and adapting them to the changing environments.

Acknowledgements

We are acknowledging the support of postgraduate students of the group "Components Based Programming" for theirhelp in analysing the projects.

References

[1] M. Fagan, Design and Code Inspections to Reduce Errors in Program Development, IBM Systems Journal, 15 (3), 1976.

[2] N.E. Fenton, Software Metrics. A Rigorous Approach, Int. Thompson Computer Press, London, 1995.

[3] M. Frentiu, H.F. Pop, The Study of Dependence of Software Attributes using Data Analysis Techniques, Studia Universitatis Babes-Bolyai, Informatica, 47 (2), 2002, 53–60.

[4] M. Frentiu, On programming style, Technical report, Babes-Bolyai University, Department of Computer Science, http://www.cs.ubbcluj.ro/ mfrentiu/articole/style.html

[5] M. Frentiu, H.F. Pop, Documents produced at the individual project, Tehnical report, Babes-Bolyai University, Department of Computer Science, http://www.cs.ubbcluj.eo/ mfrentiu/articole/project.html

[6] T. Gilb, D. Graham, Software Inspection, Addison-Wesley, 1993

[7] R.J. Hathaway, J.C. Bezdek, Fuzzy C-Means Clustering of Incomplete Data, IEEE Transactions on Systems, Man, Cybernetics, Part B: Cybernetics, 31 (5), 2001, 1062–1071.

[8] C. Ho-Stuart, R. Thomas, Laboratory Practice with Software Quality Assurance. Proc. of the 1996 International Conference on Software Engineering: Education and Practice, IEEE, 1996, 220–225.

[9] ISO 9126, Information Technology – Software Product Evaluation – Quality Characteristics and Guidelines for their Use, http://www.iso.org

[10] J.C. Knight, E.A. Myers, An Improved Inspection Technique, Comm. ACM, 36 (11), 1993, 51–61.

[11] O. Laitenberger, A Survey on Software Inspection Technologies, Handbook

[12] W.W. McMillan. What Leading Practitionners Say Should Be Emphasized in Students' Software Engineering Projects, IEEE, 1999, 177–185.
[13] M. Newby, An Empirical Study Comparing the Learning Environments of Open and Closed Computer Laboratories, Journal of Information Systems Education, 13 (4), 303–314.
[14] D.L. Parnas, A.J. van Schowen, S. Po Kwan, Evaluation of Safety-critical Software, Comm.A.c.M., 33(6), 1990, 636–648.
[15] H.F. Pop, Intelligent Systems in Classification Problems, Ph.D. Thesis, Babes-Bolyai University, Faculty of Mathematics and Computer Science, Cluj-Napoca, 1995.
[16] H.F. Pop, SAADI: Software for fuzzy clustering and related fields, Studia Universitatis Babes-Bolyai, Series Informatica 41 (1), 1996, 69–80.
[17] V.E. Veraart, S.L. Wright, Experience with a Process-driven Approach to Software Engineering Education. Proc. of the 1996 International Conference on Software Engineering: Education and Practice, IEEE, 1996, 406–413.
[18] H. Younessi, D.D. Grant, Using CMM to Evaluate Student SE projects. Proc. of the 1996 International Conference on Software Engineering: Education and Practice, IEEE, 1996, 386–391.

Babeş-Bolyai University, Faculty of Mathematics and Computer Science, RO-400084 Cluj-Napoca, Romania

E-mail address: mfrentiu@cs.ubbcluj.ro

E-mail address: ilazar@cs.ubbcluj.ro

E-mail address: hfpop@cs.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLVIII**, Number 2, 2003

COMPLEXITÉ ALGEBRIQUE DES ALGORITHMES GÉOMETRIQUES — LE PROBLÈME D'INTERSECTIONS D'UN ENSEMBLE DE SEGMENTS

RADU-LUCIAN LUPŞA

Abstract. Il est connu qu'un problème difficile dans l'implantation des algorithmes géometriques est le calcul exact des points d'intersection des droites ou des courbes, ainsi que l'évaluation des prédicats géometriques. D'un part, il s'agit du fait que, si les calcules sont inéxactes, il est possible qu'un algorithme se comporte complètement imprevisible. D'autre part, beaucoup d'algorithmes ne traitent pas explicitement les cas particuliers dégénérés.

On étudie dans cet article le cas de l'algorithme de Balaban pour trouver les intersections d'un ensemble de segments, du point de vue de l'evaluation des prédicats géometriques.

1. Introduction

L'article étudie les problèmes liés à l'evaluation des predicats et au traitement des cas dégénérés. Le cas étudié est celui de l'algorithme de Balaban pour trouver les intersections du'un ensemble de segments.

Le reste de l'introduction presente l'algorithme de Balaban [1], y compris les définition dont on a besoin à la suite. Puis, on présente l'étude de l'auteur sur les solution au problème des cas dégénérés, au problème de lévaluation des prédicats et finalement une comparaison des performances.

Quelques notations et définitions:

Si les ordonées des extremités d'un segment ne nous interrèsse pas, le segment sera noté (l, r), où l et r sont les abscises des extrémités gauche, respectivement droite.

On appelle verticale une droite verticale passant par une extrémité de segment. La verticale v est donc la droite $x = v$.

On appelle bande une partie du plan délimitée par deux verticales.

On dit qu'un segment (l, r) :

- est interieur pour la bande (b, e) si $l > b$ et $r < e$;
- traverse la bande (b, e) si $l \leq b$ et $r \geq e$.

Received by the editors: December 10, 2003.

2000 *Mathematics Subject Classification.* 68U05.

1998 *CR Categories and Descriptors.* I.3.5: [**Computer Graphics**]: Computational Geometry and Object Modeling – *Geometric algorithms, languages, and systems*.

On note $\mathrm{Int}_{b,e}(S_1, S_2)$, où S_1 et S_2 sont des ensembles quelquonques de segments l'ensemble des pairs $(s_1, s_2) \in S_1 \times S_2$ de segments qui s'intersectent dans la bande (b, e).

Une liste de segments qui intersectent une verticale v est triée par rapport à cette verticale si elle est triée dans l'ordre croissante des ordonée des points d'intersection entre les segments de la liste et la verticale v.

Enfin, on appelle un escalier un triplet (b, e, Q) où (b, e) est une bande et Q est un ensemble de segments qui traversent la bande (b, e) et ne s'intersectent pas à l'interieur de la bande (l'ensemble Q doit être trié d'après la règle ci-dessus par rapport aux verticales b et e (l'ordre est la même par rapport aux deux)).

Pour trouver les intersections à l'interieur d'une bande elementare (sans extrémitées de segments à l'interieur), on utilise l'algorithme suivant:

```
**procedure** *ChercherDansBande*($b, e, L, R$)
    *Diviser*($b, e, L, Q, L'$)
    **if** $Q = \emptyset$
        **then** $R := L$
        **else**
            trouver et ecrire $Int_{b,e}(Q, L')$
            *ChercherDansBande*($b, e, L', R'$)
            *Fusioner*($b, e, R', Q, R$)
    **endif**
**end**
```

Ici:

- *Diviser*(b, e, L, Q, L') divise la liste L triée par rapport à la verticale b dans l'escalier maximal (au sens d'inclusion d'ensembles) (b, e, Q) et la liste restante L'.
- *Fusioner*(b, e, R', Q, R) fusione la liste R' triée par rapport à la verticale e avec l'escalier (b, e, Q) en sortant la liste R triée par rapport à e.

Ainsi *ChercherDansBande* divise la liste initiale L en deux, Q et L'; maintenant

$$\mathrm{Int}_{b,e}(L, L) = \mathrm{Int}_{b,e}(L', L') \cup \mathrm{Int}_{b,e}(L', Q) \cup \mathrm{Int}_{b,e}(Q, Q)$$

où $\mathrm{Int}_{b,e}(Q, Q) = \emptyset$ et $\mathrm{Int}_{b,e}(L', L')$ est trouvé par l'appel récursiv.

Il faut remarquer aussi que *Diviser* obtient aussi les positions des segments de L' dans l'escalier Q, de maniere que les intersections d'un segment de L' avec les marches de Q puissent être trouvées en balaiant Q à partir de la position donnée par *Diviser* dans les deux sens jusqu'à trouver une marche qui n'intersecte pas le segment.

Passons maintenant à l'algorithme principal. Balaban décrit au fait deux algorithmes, dont le premier est un peu plus simple, mais asimptotiquement sub-optimal (complexité $O(n \log^2 n + k)$, où n est le nombre de segments et k le nombre d'intersections), et le deuxième est optimal ($O(n \log n + k)$) mais plus compliquée.

L'algorithme sub-optimal est le suivant

```
begin
    $L$ := {le segment qui commence sur la première verticale}
    $I$ :=l'ensemble initial moins $L$ moins le segment qui
            finit sur la derniere verticale
    *ChercherDansArbre(première vert., dernière vert.*, $L, I, R$*)*
end
```

avec:

```
procedure *ChercherDansArbre*($b, e, L, I, R$)
    if $b$ et $e$ consecutives
        then *ChercherDansBande*($b, e, L, R$)
        else
            $c$ :=la verticale qui divise en deux l'ensemble
                    des verticales comprises entre $b$ et $e$
            *Diviser*($b, e, L, Q, L_1$)
            trouver $\mathrm{Int}_{b,e}(Q, L_1)$
            $I_1 := \{s \in I | s$ interieur pour $(b, c)\}$
            *ChercherDansArbre*($b, c, L_1, I_1, R_1$)
            $s$ :=le segment dont une extremit
            if $s$ commence sur $c$
                then $L_2 := R_1 \cup \{s\}$
                else $L_2 := R_1 \setminus \{s\}$
            endif
            $I_2 := \{s \in I | s$ interieur pour $(c, e)$
            *ChercherDansArbre*($c, e, L_2, I_2, R_2$)
            trouver $\mathrm{Int}_{b,e}(Q, I)$
            *Fusioner*($b, e, R_2, Q, R$)
            trouver $\mathrm{Int}_{b,c}(Q, R_2)$
    endif
end
```

Dans cet algorithme, la seule operation qui demande un temps plus grand que $O(n \log n + k)$ est *trouver* $\mathrm{Int}_{b,e}(Q, I)$, car elle nécesite une recherche binnaire de la position de chaque segment de I dans Q. Pour éviter ceci, la solution consiste à garder les positions des segmentes de I trovée dans les appels récursives de *TreeSearch*; en ce but il faut garder des rélations entre les escaliers générés au differents niveaux d'appel de *TreeSearch*.

2. Cas particuliers et dégénérées

Le premier problème à l'implantation de cet algorithme est le fait que l'algorithme ne traite pas éxplicitement les cas particuliers:

- si l'intersection de deux segmentes se trouve exactement sur une verticale;

- si le point d'intersection se trouve sur une extremité de segment;
- si deux extremités de segment se trouvent sur la même verticale;
- si un segment est vertical;
- si deux segments se superposent (ont la même droite-support).

A ce point il est utile de regarder un peu la preuve de correction de l'algorithme. On s'apperçoit alors que les démandes sur les prédicats sont les suivantes:

(1) Si deux segments ne s'intersectent pas dans une bande, alors leur ordre relative sur les verticale gauche et droite doit être la même;
(2) Si un segment est localisé entre deux marches m_i et m_{i+1} d'un éscalier $Q_{b,e}$ et il n'intersecte ni m_i, ni m_{i+1}, alors il ne doit intersecter aucune autre marche de $Q_{b,e}$;
(3) Un segment ne peut intersecter que des marches consecutives d'un même escalier.

Prenons ces cas un à la fois.

Si le point d'intersection se trouve exactement sur une verticale, il suffit de considerer q'il se trouve "un peu à droite" ou "un peu à gauche", donc dans une des deux bandes délimitées par la verticale en question. Mettons-le donc à gauche. Maintenant pour garder la cohérence, si deux degmentes s'intersectent sur une verticale, il faut mettre en premier sur la liste triée celui qui se trouvera en premier sur la verticale suivante (car ils ne s'intersecteront plus dans la bande de droite), c'est-à-dire le premier en ordre trigonométrique dans le demi-plan droit.

Maintenant si le point d'intersection coïncide avec une extrémité, il se trouvera donc sur une verticale, et conformement au paragraphe precedent il va être consideré dans la bande de gauche. S'il coïncide avec l'extremité droite du segment, il va être traité comme s'il était à l'intereur; par contre s'il coïncide avec l'extremité gauche il ne sera pas vu car le segment n'existe pas dans la bande gauche. Ce cas d'intersection doit donc être tarité lors de l'insertion du segment sur la liste de segments correspondante à cette verticale. *Remarque:* si au moment de l'insertion l'autre segment est une marche, l'intersection sera vue normalement par l'algorithme.

S'il existe plusieurs extrémités de segment sur la même verticale, il y a deux solutions: soit on considere des bandes de longueur zero, soit on prépare la liste triée des extrémitées (avec les segments correspondants) et on la fusione avec la liste des segments associés à la verticale. J'ai pris la deuxiéme approche. En faisant ainsi, le traitement d'une verticale ne consiste plus seulement à ajouter ou effacer un segment (obtenir la liste L_2 à partir de la liste R_1), mais à fusioner la liste L_1 avec la liste des segments qui commencent sur la verticale traitée et a effacer par la même occasion les segments qui finissent sur la verticale traitée. En même temps, on garde une liste de segments verticaux, qu'on met au jour chaque fois qu'on rencontre une extrémité d'un segment vertical.

Enfin, le dernier problème concerne le cas où deux segments partagent la même droite support et leur intersection est un segment. Pour les segments verticaux, ceci n'est pas dérangeant, leur intersection etant trouvée lors du traitement de la verticale sur laquelle ils se trouvent. Si, au contraire, les segments sont obliques ou horizontals, ils s'intersectent dans plusieurs bandes en agrandisant le nombre d'intersections et donc le temps de calcule. (n segmentes superposés peuvent donner ainsi jusqu'à $\frac{1}{6}(2n^3 + n)$ “points” d'intersection). Dans ce cas on peut considerer qu'il s'intersectent toujours en un seul point; soit ce point le point le plus à droite de l'intersection. L'ordre de ces segments sur une liste associée à une verticale n'a pas alors d'importance.

3. Evaluation des prédicats

Il est connu que les algorithmes géométriques sont très sensibles aux erreurs numériques; d'ici la nécessité de faire les calcules exactes, ou au moins que les valeurs de vérité des divers predicats évalués pendant l'execution de l'algorithm soit cohérentes; sinon il y a le risque que la réponse de l'algorithme ne soit ni même une approximation du résultat reel, ou même que l'algorithme ne s'arrête pas.

Les calcules exactes d'autre part sont chères car non supportés directement par le materiel de l'ordinateur. Et dans tous les cas, plus longue soit la réprésentation, plus long est le temps de calcule. L'opération la plus chère en terme de démande de chiffres significatives sur la réprésentation etant la multiplication, on voit l'interrêt de réduire le plus possible le dégrée des polinômes qui apparaissent dans l'evaluation des prédicates.

Dans le cas de l'algorithme de Balaban, les prédicates décrits dans la section precedante sont basée sur les trois suivants ($<_x$ signifie “l'abscise du premier point est plus petite que celle du deuxième et $<_y$ signifie la même chose sur l'ordonée):

(1) étant donées 2 points, p_1 et p_2, est-ce que $p_1 <_x p_2$?
(2) étant donées 3 points, est-ce que p_3 se trouve à gauche, sur ou à droite de la droite p_1p_2 (orientée de p_1 vers p_2)
(3) étand donées 5 points $p_0 \dots p_4$ tels que $\{q\} = p_1p_2 \cap p_3p_4$, est-ce que $p_0 <_x q$ (q est inconnu et il n'est pas nécessaire de le connaître éxplicitement)?

En supposant une réprésentation cartesienne des points, l'évaluation des trois prédicats ci-desus est equivalente á trouver le signe d'un polynôme de degrée 1,2 respectivement 3, irreductible. Pour évaluer ledit signe, il faut en principe travailler avec des nombres ayant 3 fois plus de chiffres que les coordonées des points. Mais dans [2] il est montré que l'evaluation “presque exacte” du predicate 3 est possible en travaillant en réels (IEEE 754 [5]) double-précision, les coordonées des points étant simple-précision. Ce “presque exacte” veut dire que si le signe calculé du polynôme est -1 alors le signe réel est aussi -1; même chose pour $+1$; mais si le signe calculée est 0 alors on ne sait rien sur le signe réel.

4. Les performances

Dans le tableau ci-dessous sont marqués les temps de calcul utilisés (sur le même ordinateur) par l'algorithme naïf et l'algorithme sub-optimal et l'algorithme optimal de Balaban en utilisant chacun les predicates géométriques de la bibliothèque

CGAL [3] (qui utilisent a leur tout les rationnels en précision illimitée de la bibliotheque LEDA [4]) et des predicates écrits en utilisant les réels double-précision.

Les exemples de test ont été generés soit aléatoirement, soit (pour les dernières 4) sous la forme d'une grille carrée.

Méthode génération	Nr. seg.	Nr. pairs	Temps naïf	Temps sub-opt CGAL	Temps opt CGAL	Temps sub-opt dbl-prec	Temps opt dbl-prec
aléat	100	215	1	1	4	1	1
aléat	200	886	3	3	11	1	2
aléat	800	14113	45	29	97	8	24
aléat	2000	6869	—	28	129	7	29
aléat	5000	43184	—	122	550	28	125
aléat	2000	89352	—			47	99
aléat	5000	555640	—			261	382
grille	800	2281	15	1	1	1	1
grille	1800	5221	84	1	1	1	1
grille	5000	14701	—	2	2	4	2
grille	20000	59401	—	9	8	16	8

Table 1. Temps d'execution (en secondes) pour (en l'ordre) l'algorithme naïf, l'algorithme sous-optimal avec les prédicats implanté par CGAL, l'algorithme optimal avec les prédicats CGAL, l'algorithme optimal avec les prédicats évalués en double-précision et l'algorithme optimal avec les prédicats en double precision

On voit facilement que pour des exemples de taille raisonnable, l'algorithme sous-optimal asimptotiquement se comporte nettement mieux, le gain d'un facteur $\log n$ (7–13 dans les exemples ci-dessus) étant contrebalancé par la perte due aux complications de l'algorithme. Il faut remarquer aussi le gain de vitesse obtenu en remplaceant les rationnels en précision illimité (bibliothéque LEDA) par les réels double-precision de la machine.

References

[1] I. Balaban, An Optimal Algorithm for Finding Segment Intersections, Proceedings of the Eleventh Annual Symposium on Computational Geometry, Vancouver, Canada, June 5-7, 1995, pg. 211-219

[2] Jean-Daniel Boissonnat, Franco P. Preparata, Robust Plane Sweep for Intersecting Segments, Rapport de recherche no. 3270 septembre 1997, Institut National de Recherche en Informatique et en Automatique

[3] `http://www.cgal.org/`

[4] `http://www.mpi-sb.mpg.de/LEDA/leda.html`

[5] `http://grouper.ieee.org/groups/754/`

Babeş-Bolyai University, Faculty of Mathematics and Computer Science, RO-400084 Cluj-Napoca, Romania

E-mail address: `rlupsa@cs.ubbcluj.ro`

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLVIII**, Number 2, 2003

PROFESSOR DOINA TĂTAR AT HER SIXTIES

MILITON FRENŢIU

Professor Doina Tătar was born on April 26th, 1943 in Curtea de Argeş, Argeş County. After finishing secondary school in Sibiu, in 1961, she studied at the Babeş-Bolyai University of Cluj. She graduated in Mathematics in 1966.

Immediately after graduation she was named assistant at the University of Craiova, in the Electronics Department. She worked at this university until 1991. In 1990 she became assistant professor at this University. In 1991 she moved in Cluj-Napoca, where she got a position at the Babes-Bolyai University, Department of Computer Science.

Professor Doina Tatar obtained her PhD in Mathematics in 1975, at University of Bucharest, under the supervision of Prof. Dr. Paul Constantinescu.

She has a succesful activity in Computer Science. Her main interests are the Foundamentals of Computer Science, Formal Languages, Automata Theory, Logics and Automated Proof Theory, Rewriting Systems, Computational Linquistics. She published 62 researh papers and two books, as can be seen in the appendices.

Prof. Tătar has introduced in the curricula the lectures on "Natural Language Processing" and "Automated Theorems Proving and Rewriting Systems". The lectures taught by Prof. Tătar are characterised by a high scientific level, being at the world level of these fields, and they have an important contribution to the didactic process. She has supervised a large number of B.Sc. and Master theses, as well as the students scientific research activity. Many of her former disciples have successfully continued doctoral studies abroad, a few of them being today faculty members or researchers with their universities.

So, she is remarked as an eminent scientist and pedagogue, at the same time, with true love and devotion for her students.

Now, on celebrating her 60th birthday, we wish her Many Happy Returns of the Day and a long life in health and happiness.

Received by the editors: October 15, 2003.

Appendices

1. Papers published in jurnals

(1) Tătar Doina: “Entropia-criteriu cantitativ al stilului”, Studii şi cercetări lingvistice”, nr 6, 1967, pp 221–231.
(2) Belea Constantin, Tătar Doina: “Sinteza optimală neliniară a unor sisteme de conducere automată cu obiective liniare”, Analele Universităţii din Craiova, 1970, nr. 1, pp 191-203.
(3) Tătar Doina: “Câteva algoritme de optimizare la nivel operativ”, Analele Universităţii din Craiova, 1976, nr 4, pp 49–53.
(4) Tătar Doina: “Metodologia sistemului informatic teritorial al zonei Amaradia”, Analele Universităţii din Craiova, 1976, nr 1, pp50-56.
(5) Tătar Doina: “Asupra cercetării şi prelucrării automate a informaţiei privind populaţia”, Analele Universităţii din Craiova, 1976, nr 1, pp56-64.
(6) Tătar Doina: “Prelucrarea automată a datelor de fundamentare a studiului privind recrutarea forţei de muncă”, Analele Universităţii din Craiova, 1981, nr 12, pp80-84.
(7) Tătar Doina: “Utilizarea schemelor de recursie in structurarea programelor”, Buletinul Universităţii din Braşov, 1981, seria C, vol.XXIII, pp13-20.
(8) Tătar Doina: “Program schemes and associated languages”, Bulletin Mathematique de la Societe Math. R. S. Roumanie, 29(77) (1985), no. 2, 161–166.
(9) Tutunaru Marcela, Mihăilescu Ion, Tătar Doina: “Change of the nature of skin absorbtion during CO_2 -laser beam irradiation of a metallic target”, Optica applicata, Belgia, vol 16, nr 3, 1986, pp209-214.
(10) Tătar Doina, Tătar Liviu, Tutunaru Marcela: ”Calcularea coeficientului de absorbţie a radiaţiei laser”, Analele Universităţii din Craiova, 1984, nr 2, pp88-94.
(11) Tătar Doina: “Some remarks about the theory of transformational grammars”, Revue roumaine de linguistique, tom XXXII, nr 2, 1987, pp137-141.
(12) Tătar Doina: “Equivalent criteria for recursive programs”, Studii şi Cercetări Matematice, 38 (1986), no. 1, pp72-78.
(13) Tătar Doina: “Asupra schemelor monadice”, Studii şi Cercetări Matematice, 39 (1987), no. 5, pp455-461.
(14) Tătar Doina: “Using a syntactic grammar associated with a recursive scheme in the study of programs”, Studii şi Cercetări Matematice, 40 (1988), no. 4, pp337–347.

(15) Tătar Doina: "Normalized rewriting systems and applications in the theory of programs", Analele Univ. Bucuresti, Seria Matematica, 1989, no. 2, pp76–80.
(16) Tătar Doina: "A new method for the proof of theorems", Studia Universitatis "Babes-Bolyai", Seria Mathematica, 36 (1991), no. 3, pp83–95.
(17) Tătar Doina: "Logical state transition systems as finite automata", Science Research seminars, Seminar on Computer Science, preprint 5, 1992.
(18) Tătar Doina: "Term rewriting systems and completion theorems proving", Studia Universitatis "Babes-Bolyai", Seria Mathematica, 1992, pp117–127.
(19) Tătar Doina, Lupea Mihaela: "A note on non-monotonic logics", Studia Universitatis "Babes-Bolyai", Seria Mathematica, nr3, 1993, pp109-115.
(20) Tătar Doina: "Logical grammars and unfold transformations of logic programming", Studia Universitatis "Babes-Bolyai", Seria Mathematica, nr3, 1994, pp75-83.
(21) Tătar Doina: "Logical grammars as a tool for studying Logic Programming", Studia Universitatis "Babes-Bolyai", Seria Mathematica, nr.3, 1994, pp 83-95.
(22) Dumitrescu Dan, Tătar Doina: "Normal forms of fuzzy formulas and their minimization", Fuzzy Sets and Systems 64 (1994), nr.1, 113–117.
(23) Tătar Doina: "Automated theorem proving with semantic tableaux", Research seminar in Computer Science, Preprint nr. 2, 1995, pp 27-36.
(24) Tătar Doina: "Buchberger algorithm in theorem proving", Analele Universităţii "A.I.Cuza", seria Informatica, 1995, pp85-92.
(25) Tătar Doina: "Unification based and orient-object based approaches to grammars", Research seminar in Computer science, nr2, 1996, pp63-74.
(26) Tătar Doina: "Logical grammar as grammatical view of logic programming", Analele Universităţii Bucuresti, Seria mat-inf, XLV, 1996, 2, pp3-12.
(27) Tătar Doina: "Compiling definite clause grammars", Studia Universitatis " Babes-Bolyai", Seria Informatica, vol I, nr1, 1996, pp45-56.
(28) Tătar Doina, Varga Viorica: "Simplification of magic-set rules by logic grammars", Studia Universitatis "Babes-Bolyai", seria Informatica, vol I, nr2, 1996, pp 19-30.
(29) Tătar Doina: "Feature structures in NLP", Research Seminar in Computer Science, nr. 3, 1998, pp93-104.
(30) Tătar Doina, Dumitrescu Adina: "Reasoning with frame - based and object - oriented knowledge", Studia Universitatis "Babes-Bolyai", Seria Informatica, 1997, nr.2, pp 13-24.
(31) Tătar Doina, Câmpan Sorana: "Incomplete databases and constraint logic programming", Studia Universitatis "Babes-Bolyai", Seria Informatica, 1997, nr1, pp63-77.

(32) Tătar Doina, Lupea Mihaela: "Indexed grammars and unification grammars", Studia Universitatis "Babes-Bolyai", Seria Informatica, 1998, nr 1, pp39-46.
(33) Tătar Doina, Şerban Gabriela: "Well-typedness in logic programming with types", Studia Universitatis "Babes-Bolyai", Seria Informatica, 1998, nr.2, pp27-36.
(34) Tătar Doina: "Unification grammars in natural language processing", in "Recent topics in mathematical and computational linguistics", Editors: C. Martin-Vide, G. Paun, Ed. Academiei, 2000, pp289-300.
(35) Tătar Doina, Oltean Mihai: "Theorem proving and DNA computing", Studia Universitatis "Babes-Bolyai", Seria Informatica, 1999, nr 1, pp62-72.
(36) Oneţ Adrian, Tătar Doina: "Semantic representation of quantitative natural language sentences", Studia Universitatis "Babes-Bolyai", Seria Informatica, 1999, nr 2, pp99-109.
(37) Oneţ Adrian, Tătar Doina: "Order-sorted logic for knowledge representation with application in NLP", Research Seminar on Computer Science, 2000, pp75-84
(38) Oneţ Adrian, Tătar Doina: "The semantic representation of NL sentences. A theoretical and practical approach", Bulletin for Applied and Computing Mathematics, ISBN 0133-3526, vol XCIII, 2001, pp195-204.
(39) Duda Adrian, Şerban Gabriela, Tătar Doina: "Training Probabilistic Context-Free Grammars as Hidden Markov Models", Studia Universitatis "Babes-Bolyai", Seria Informatica, 2000, nr 1, pp17-30.
(40) Tătar Doina, Avram Dana: "Phrase Generation in Lexical Functional Grammars and Unification Grammars", Studia Universitatis "Babes-Bolyai", Seria Informatica, 2000, nr 1, pp69-78.
(41) Oneţ Adrian, Tătar Doina: "Semantic Analysis in Dialog Interfaces", Studia Universitatis "Babes-Bolyai", Seria Informatica, 2000, nr 1, pp79-88.
(42) Tătar Doina, Şerban Gabriela: "Term rewriting systems in logic programming and functional programming", Studia Universitatis "Babes-Bolyai", Seria Informatica, 2000, nr 2, pp87-94.
(43) Tătar Doina: "Feature Structures in NL interfaces", Analele Univ Timisoara, vol XXXVIII, nr 2, 2000, pp179-191
(44) Oneţ Adrian, Tătar Doina: " Intensional Logic Translation for Quantitative Natural Language Sentences", Studia Universitatis "Babes-Bolyai", Seria Informatica, 2001, nr 1, pp41-54.
(45) Tătar Doina, Şerban Gabriela: "A new algorithm for word sense disambiguation", Studia Universitatis "Babes-Bolyai", Seria Informatica, 2001, nr 2, pp99-108.

(46) Şerban Gabriela, Tătar Doina: "A Word Sense Disambiguation Experiment for Romanian Language", Studia Universitatis "Babes-Bolyai", Seria Informatica, 2002, nr 2, pp37-43

(47) Tătar Doina, Şerban Gabriela: "Word clustering in QA systems", Studia Universitatis "Babes-Bolyai", Seria Informatica, 2003, nr 1, pp 23-33

(48) Şerban Gabriela, Tătar Doina: "An improved algorithm on word sense disambiguation", Advances in Soft Computing, Ed. Springer, "Inteligent Information Processing and Web Mining", Editors M.A.Klopotek, S. Wierzchon, K. Trojanowski, pp 199-209.

(49) Avram Lupşa Dana, Şerban Gabriela, Tătar Doina: "From noun clusters to taxonomies on untagged corpora", Mathematical Preprint Server as *MPS, Applied Mathematics, 0309004, 12 sept. 2003*

(50) Avram Lupşa Dana, Şerban Gabriela, Tătar Doina: "Hierarhical clustering algorithms for repeating similarity values", Studia Universitatis "Babes-Bolyai", seria Informatica, 48, 2, 2003, 65–76.

2. Papers published in proceedings of conferences

(1) Tătar Doina: "Descompunerea grafurilor in raport cu operaţiile algebrice", Lucrările colocviului de matematici aplicate, noiembrie 1972, pp 197-206.

(2) Tătar Doina: "Ordonanţare prin programare liniară", Lucrările colocviului de matematici aplicate, noiembrie 1972, pp 207-215.

(3) Tătar Doina: "Optimizarea programelor prin intermediul schemelor recursive", Primul Simpozion naţional de teoria sistemelor, Craiova, mai 1980, pp253-260.

(4) Tătar Doina: "Scheme program şi limbaje asociate", Al doilea simpozion national de teoria sistemelor, Craiova, mai 1982, vol. III, pp 321-332.

(5) Tătar Doina: "Limbaje echivalente şi scheme program", Colocviul de teoria probabilităţilor şi cercetări operaţionale, Craiova, Noiembrie 1982, pp 189-197.

(6) Tătar Doina: "Notă asupra sintezei automatelor finite", Simpozionul "Odobleja", Universitatea din Craiova, 1982, pp 49-53.

(7) Tătar Doina: "Gramatici utilizate in teoria compilarii", Simpozionul "Tehnici moderne de calcul in economie", 16-17 mai 1986, pp 88-91.

(8) Tătar Doina: "Aplicatii ale limbajelor formale in teoria programelor", Simpozionul "Tehnici moderne de calcul in economie", 16-17 mai 1986, pp 91-94.

(9) Tătar Doina: "Proprietati de punct fix ale schemelor recursive şi ecuatii de limbaje", Primul Colocviu naţional de limbaje, logică şi lingvistică matematică, Braşov, 5-7 iunie, 1986, pp 209-215.

(10) Tătar Doina: "Condiţii de terminare in teoria schemelor recursive", Simpozionul National INFO - Iasi, 9-10 octombrie 1987, pp 68-74.

(11) Tătar Doina: “Utilizarea gramaticii sintactice asociată unei scheme recursive”, Simpozionul de utilizarea metodelor şi tehnicilor moderne de calcul in economie, Craiova, 9-10 octombrie, 1987, pp 68-74.
(12) Tătar Doina: “Derecursivarea programelor prin unificări in sisteme de rescriere”, Al doilea Colocviu naţional de limbaje, logică şi lingvistică matematică, Braşov, 2-3 iunie, 1988,pp225-235.
(13) Tătar Doina: “ A new method for the proof of theorems”, Al treilea Colocviu naţional de limbaje, logică şi lingvistică matematică, Braşov, 23-25 mai, 1991, pp 111-121.
(14) Tătar Doina: “Buchberger algorithm in theorem proving”, Proceedings of ROSYCS’93, 12-13 nov, Iaşi, pp490-501.
(15) Tătar Doina: “Attribute grammars and Logic programming with types”, Proceedings ROSYCS’96, Iasi, mai 1996, pp 57-69.
(16) Dumitrescu Dan, Tatar Doina, Mureşan Leila, Dumitrescu Adina: “A class of residuated lattices connected with fuzzy set theory”, Proceedings of IFSA’97, Praga, 1997, pp227-231.
(17) Tătar Doina, Dumitrescu Adina: “Reasoning with frame-based and object-oriented knowledge”, Acceptată la conferinţa SEKE’97 (Software engineering and knowledge engineering), June 18-20,1997, Madrid, Spania.
(18) Tătar Doina, Zaiu Diana: “Unification based and object-oriented based grammars”, Proceedings of LACL’97 (Logical aspects of Computational Linguistics), pp 65-70, Nancy, France, 22-24 sept 1997.
(19) Tătar Doina, Zaiu Diana: “Feature structures in NLP and object-oriented logic”, Proceedings of International Conference SPECOM’97, Cluj-Napoca, 27-30 oct 1997, pp 31-37.
(20) Tătar Doina: “Feature structures in NL interfaces”, Proceedings of International Conference SYNASC 2000, 4-6 oct.2000, Univ. Timisoara - RISC-Linz, pp82-86.
(21) Şerban Gabriela, Tătar Doina: “Word Sense Disambiguation for Untagged Corpus: Application to Romanian Language”, Lecture Notes in Computer Science nr 2588, Ed. Springer, pp 270-275, CICLing-2003, Fourth International Conference on Intelligent Text Processing and Computational Linguistics, February, 2003, Mexico City, Mexico.
(22) Tătar Doina: “Feature structures in NL interfaces”, SYNASC 2000, Timisoara, 4-6 oct. 2000. Proceedings of SYNASC 2000, International Workshop on Symbolic and Numeric Algorithms on Scientific Computing, Univ. of the West- RISC -Johannes Kepler University LINZ, pp 82-85.
(23) Şerban Gabriela, Tătar Doina: “Word Sense Disambiguation for Untagged Corpus: Application to Romanian Language”, CICLing-2003,

International Conference on Intelligent Text Processing and Computational Linguistics, February, 2003, Mexico, pp 270-275.

(24) Oraşan Constantin, Tătar Doina, Şerban Gabriela, Avram Dana, Oneţ Adrian: "How to build a QA system in your back-garden: application for Romanian", EACL, 12-17 April 2003, Budapest, pp 139-142.

(25) Şerban Gabriela, Tătar Doina: "An improved algorithm on word sense disambiguation", Proceedings of IIS 2003, Zakopane, Polland, June 2-5, 2003, pp 199-209.

(26) Avram Lupşa Dana, Şerban Gabriela, Tătar Doina: "From noun clusters to taxonomies on untagged corpora", Preprint in Computer Science, Dept. of CS, University "Babes-Bolyai", 2003, in curs de apariţie.

(27) A. Oneţ, D. Tătar: "Automated syntactic parse for English Language", trimisă la conferinţa LREC (Languages REsourses and Computation), Barcelona, mai 2003.

(28) R. Mihalcea, V. Năstase, D. Tătar: "A Web-based collaboration framework for building multilingual semantic networks", trimisă la LREC, Barcelona, mai 2003.

(29) D. Lupşa, G. Şerban, C. Orăşan, D. Tătar: "Comparative evaluation of automatically derived hierarchies for a Romanian QA system", trimisă la LREC, Barcelona, mai 2003.

3. Papers presented at national or international conferences

(1) Tătar Doina: "Câteva probleme ale algoritmizării procedeelor de calcul pentru calculatoare medii şi mici", Sesiunea cadrelor didactice de la Universitatea din Craiova, mai 1968.

(2) Tătar Doina: "Unele aspecte ale conducerii optimale dupa principiul maximului", Sesiunea cadrelor didactice de la Universitatea din Craiova, mai 1968.

(3) Tătar Doina: "Utilizarea retelelor de transport in rezolvarea problemelor de programare liniara", Sesiunea cadrelor didactice de la Universitatea din Craiova, mai 1970.

(4) Tătar Doina: "Aplicarea teoriei distribuţiei in studiul sistemelor liniare", Sesiunea cadrelor didactice de la Universitatea din Craiova, mai 1971.

(5) Tătar Doina: "Asupra unei probleme de ordonanţarea producţiei", Simpozionul de utilizarea metodelor şi tehnicilor moderne de calcul in economie, Craiova, noiembrie, 1973.

(6) Tătar Doina: "Proprietăţi cibernetice ale sistemelor economice", Colocviul de matematici aplicate, Craiova, noiembrie 1974.

(7) Tătar Doina: "Limbaje formale dependente şi independente de context", Sesiunea cadrelor didactice de la Universitatea din Craiova, noiembrie 1976.

(8) Tătar Doina: " Posibilitatea de diagnostic automat in endocrinologie", Simpozion de integrarea antropologiei şi geneticii medicale in soluţionarea problemelor de demografie teritoriala, Craiova, mai 1975.
(9) Tătar Doina: "Asupra structurii unei baze de date de antropologie", Simpozion de Integrarea antropologiei şi geneticii medicale in soluţionarea problemelor de demografie teritorială, Craiova, mai 1975.
(10) Tătar Doina: "Posibilităţi de optimizare a programelor", Colocviul de matematici aplicate, Craiova, octombrie 1980.
(11) Tătar Doina: "Criterii echivalente pentru programe recursive", Simpozionul de organizare ştiinţifică a activităţii economice, Craiova, mai 1982.
(12) Tătar Doina: "Sisteme de rescriere şi demonstrarea automata a teoremelor", Zilele Academice Clujene, "Informatica şi aplicaţiile sale", Cluj-Napoca, mai 1992.
(13) Tătar Doina: "Groebner basis in the proof of theorems", Simpozionul national ROSYCS'93, Iasi, 12-13 nov 1993.
(14) Tătar Doina: "Teste standard in logica predicatelor", Zilele Academice Clujene, "Informatica şi aplicaţiile sale", Cluj-Napoca, octombrie 1993.
(15) Tătar Doina: "Inferenţa tipurilor in programarea logică", Zilele Academice Clujene, "Informatica şi aplicatiile sale", Cluj-Napoca, octombrie 1995.
(16) Tătar Doina: "Buchberger algorithm in theorem proving", Simpozionul naţional ROSYCS'95, Iasi, nov 1995.
(17) Tătar Doina: "Automated theorem proving with semantic tableaux", International Conference PAP'96 (Practical Application of Prolog), Londra, 26-28 aprilie, 1996.
(18) Tătar Doina: "Semantica programării logice nonstandard", Al doilea Simpozion Naţional al stiintelor cognitive, 2-4 mai 1996, Universitatea "Babes-Bolyai", Cluj-Napoca.
(19) Tătar Doina, Zaiu Diana: "Unification based and object-oriented based grammars", LACL'97 (Logical aspects of Computational Linguistic), Nancy, France, 22-24 sept,1997.
(20) Tătar Doina: "Unificare şi constrângeri in prelucrarea limbajului natural", Zilele Academice Clujene, "Informatica şi aplicaţiile sale", Cluj-Napoca, 10 iunie, 1998.
(21) Tătar Doina, Oneţ Adrian: "Automated definite clause grammara compiling", Simpozionul national ROSYCS'98, Iasi, 28-29 mai 1998

4. Books

(1) Tătar Doina: "Inteligenţă artificială: demonstrarea automată, prelucrarea limbajului natural", Editura Albastră, Microinformatica, 2001, ISBN 973-9443-99-0, 230 pg.

(2) Tătar Doina: "Inteligenţă artificială. Aplicaţii in prelucrarea limbajului natural", Editura Albastră, Microinformatica, 2003, ISBN 973-650-100-0, 249 pg.

5. Manuals

(1) Tătar Doina, Tutunea Ion: "Bazele Informaticii I", Reprografia Universităţii din Craiova, 1978, 178 pages.
(2) Tătar Doina, Bereanu Constantin: " Caiet de lucrari in limbajele Basic şi Fortran", Reprografia Universităţii din Craiova, 1979, 120 pages.
(3) Tătar Doina: "Bazele Informaticii II.", Reprografia Universităţii din Craiova, 1981, 175 pages.
(4) Tătar Doina: "Curs de Bazele Informaticii.", Reprografia Universităţii din Craiova, 1988, 185 pages.
(5) Tătar Doina: "Bazele matematice ale calculatoarelor", Reprografia Universităţii "Babes-Bolyai", 1993, 190 pages.
(6) Tătar Doina: "Bazele matematice ale calculatoarelor", Reprografia Universităţii "Babes-Bolyai", 1999, 196 pages.
(7) Tătar Doina: "Bazele matematice ale calculatoarelor", Reprografia Universităţii "Babes-Bolyai", 2004, to appear.

Babeş-Bolyai University, Faculty of Mathematics and Computer Science, RO-400084 Cluj-Napoca, Romania
E-mail address: mfrentiu@cs.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLIX**, Number 1, 2004

APOLOGY ON PLAGIARISM PAPERS

THE EDITORS

Since the preceding issue has been send to print we have found out, and have been informed by more interested readers that the following papers are plagiates:

- D. Marcu, *The Chromatic Number of Triangle-Free Regular Graphs*, Studia Universitatis Babeş-Bolyai Series Informatica, 47 (1), 2002, p. 54–56.
- D. Marcu, *A Note on the Chromatic Number of a Graph*, Studia Universitatis Babeş-Bolyai Series Informatica, 47 (2), 2002, p. 105–106.
- D. Marcu, *A Note on the Chromatic and Independence Number of a Graph*, Studia Universitatis Babeş-Bolyai Series Informatica, 48 (2), 2003, p. 11–16.

According to practices currently in place, these papers have been reviewed, as always, by a panel of two experts. They have made all possible effort to ensure the scientific quality and accuracy of the papers submitted to the journal. However, we are not always able to verify the originality of every paper submitted, and, as usually, this rests with the responsibility of the author.

After a careful consideration, we have decided to retract the papers under scrutiny; the papers will be marked as such on the journal web page. As we have lost the confidence in Mr. Dănuţ Marcu, the author of these plagiates, we have decided to ban Mr. Marcu from publishing in our journal.

We are apologizing to the international scientific community for this situation. Despite this, we are ensuring our readers that we are continually working to ensure a high scientific quality for our journal. As such, they may continue to consider our journal as the journal of their choice.

The Editors

Faculty of Mathematics and Computer Science, Department of Computer Science, Babes-Bolyai University, 400084 Cluj-Napoca, Romania
E-mail address: `studia-i@cs.ubbcluj.ro`

Received by the editors: May 15, 2004.