Anul XLVI 2001

# S T U D I A
# UNIVERSITATIS BABEŞ-BOLYAI

# INFORMATICA

# 2

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

**SUMAR – CONTENTS – SOMMAIRE**



**RECENZII – REVIEWS – ANALYSES**

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLVI**, Number 2, 2001

# CIRCULAR FITTINGS FOR FINITE PLANAR POINT SETS

GHEORGHE CIMOCA

Abstract. Using the power of a point with respect to a circle, a (hopefully new) circular fitting for a finite point set in $\mathbb{R}^2$ is proposed. Finally, necessary and sufficient conditions for the collinearity and, respectively the concyclicity of $n$ ( $n \geq 2$, respectively $n \geq 3$ ) distinct points in $\mathbb{R}^2$ are given.

## 1. Introduction

Twenty five years ago, while the writing of [6] was in progress, we needed an algorithm to approximate a finite planar point set by an arc of a circle. The generated points were located on a certain area (specifically the foot) of a gear-tooth profile. At that moment we were not able to find any specific reference, but using the least squares method we quickly fitted a first model, and we later derived a second model as a minimax approximation problem [2]. The resulting algorithms were good enough for the specific purpose. However, the uniqueness of the solution was not assured because of the non-linearity of the normal equations of the first model, respectively the non-linear non-convex programming problem arisen in the second approach.

More recently, while designing a special disc milling cutter [1], we encountered the same approximation problem. This time, the uniqueness of the approximation circle was essential. Consequently, we returned to the old attempts and finally, by figuring out a new approach, we arrived at a simple and, simultaneously, very suitable solution. The result might be known, although we haven't found it mentioned anywhere, not even in the regression "Bible" [4].

In the subsequent paragraph we shall present both the non-linear fit models [2] and the new circular approximation technique for finite planar point sets.

---

2000 *Mathematics Subject Classification.* 65D10, 65J02.

1998 *CR Categories and Descriptors.* I.3.5[**Computational Geometry** and **Object Modelling**].

## 2. Circular fittings of finite point sets in $\mathbb{R}^2$

Let $S$ be a finite set of points in the Euclidean plane $\mathbb{R}^2$. Let us assume that $S$ contains at least 3 points, not all of them being collinear.

Our problem is to select a point $C(x, y) \in \mathbb{R}^2$ as the center of a circle, and a positive number $z$ as its radius, so that this circle best approximates the considered point set $S$.

Related to our subject is the "minimum spanning circle" problem, in which the smallest circle that encloses a given set of points is sought [5]. The smallest enclosing circle is unique and is either the circumcircle of three extremal points of the convex hull of the given point set, $CH(S)$, or is defined by two of them as a diameter. Hence, the algorithm based on the previous remark examines only the extremal points of $CH(S)$.

On the other hand, the "largest empty circle" problem (i.e., finding the largest circle which contains no points of the set $S$) has neither a bounded nor a unique solution. In order to avoid unboundedness, a certain restriction regarding the center should be imposed. For instance, $C \in CH(S)$ is presented in [5] as a "necessary" condition, which is excessive and inaccurate. Restricting $C$ so that it belongs to any arbitrarily given compact subset of $\mathbb{R}^2$ would suffice. Anyway, the uniqueness can't be guaranteed.

Such problems frequently arise in industrial engineering — e.g., the siting of emergency facilities (police stations, hospitals, &c.), the location of radio transmitters, the positioning of a source of pollution or of a new business that is not going to compete for territory with established rival outlets.

In contrast to the above-mentioned proximity (covers or gaps) problems, we shall consider all the points of $S$ by means of their "distances" to the circle in question. The first two approximation models use the Euclidean distance from a point to a circle, measured on the normal direction, whereas for the third model the distance is replaced by the power of the point with respect to the circle. In fact, these quantities are estimates of the errors.

Let $S = \{P_i(a_i, b_i) \in \mathbb{R}^2 \ ; \ i = 1, \ \ldots, n\}$, where $3 \leq n < \infty$, $P_i \neq P_j$, for $i \neq j$, $i, j = 1, \ \ldots, n$, not all $P_i$ being collinear.

**Circular fit model I, using the least squares method.** Minimize:

$$\sum_{i=1}^{n} [\sqrt{(x - a_i)^2 + (y - b_i)^2} - z]^2$$

in respect with the real variables $x, y, z$, where $0 \leq z \leq M$, $M$ being a technological restriction.

The normal equations are:

$$\sum_{i=1}^{n}(x-a_i) - z\cdot\sum_{i=1}^{n}\frac{x-a_i}{\sqrt{(x-a_i)^2+(y-b_i)^2}} = 0,$$

$$\sum_{i=1}^{n}(y-b_i) - z\cdot\sum_{i=1}^{n}\frac{y-b_i}{\sqrt{(x-a_i)^2+(y-b_i)^2}} = 0,$$

$$n\cdot z - \sum_{i=1}^{n}\sqrt{(x-a_i)^2+(y-b_i)^2} = 0.$$

Substituting

$$z = \frac{1}{n}\cdot\sum_{j=1}^{n}\sqrt{(x-a_j)^2+(y-b_j)^2}$$

we obtain the following nonlinear system:

$$\sum_{i=1}^{n}\{(x-a_i)[n\cdot D_i(x,y) - \sum_{j=1}^{n}D_j(x,y)]\} = 0,$$

$$\sum_{i=1}^{n}\{(y-b_i)[n\cdot D_i(x,y) - \sum_{j=1}^{n}D_j(x,y)]\} = 0,$$

where $D_i^2(x,y) = (x-a_i)^2+(y-b_i)^2$.

Taking the barycenter of $S$ as the initial approximation, the well-known Newton-Raphson algorithm [3] works quite well. Some remarks concerning the circle uniqueness appear in [2].

**Circular fit model II, using the Chebyshev norm.** Minimize:

$$\max\{\left|\sqrt{(x-a_i)^2+(y-b_i)^2}-z\right|, i=1,\ldots,n\}$$

in respect with the real variables $x, y, z$, where $0\le z\le M$.

Introducing a new variable $u$, the above problem can be reformulated as the following mathematical programming problem:

Minimize $u$
subject to:

$$x^2+y^2-z^2-u^2-2uz-2xa_i-2yb_i+a_i^2+b_i^2 \le 0,\ \ i=1,\ldots,n,$$

$$x^2+y^2-z^2-u^2+2uz-2xa_i-2yb_i+a_i^2+b_i^2 \ge 0,\ \ i=1,\ldots,n,$$

$$x\in\mathbb{R},\ \ y\in\mathbb{R},\ \ 0\le z\le M,\ \ 0\le u\le\varepsilon$$

where $\varepsilon$ is a maximum admissible error.

More details on this method, called the Megiddo method, can be found in [5].

Now let's remember that the number $\rho = d^2 - r^2$, where $d$ is the distance from a point $P \in \mathbb{R}^2$ to the center $C$ of a circle of radius $r$, denoted by $(C, r)$, is called the power of the point $P$ with respect to the circle. For any point lying outside the circle $(C, r)$ we have the inequality $\rho > 0$, and for all points lying inside that circle we have $\rho < 0$. If $P \in (C, r)$ we obviously have $\rho = 0$.

Using the power of a point with respect to a circle we can derive:

**Circular fit model III.** Minimize:

$$\sum_{i=1}^{n}[(x-a_i)^2 + (y-b_i)^2 - z^2]^2,$$

in respect with the real variables $x$, $y$, $z$, where $0 < z \leq M$.

For this problem, the normal equations are:

$$\sum_{i=1}^{n}(x-a_i)[(x-a_i)^2 + (y-b_i)^2 - z^2] = 0,$$
$$\sum_{i=1}^{n}(y-b_i)[(x-a_i)^2 + (y-b_i)^2 - z^2] = 0,$$
$$\sum_{i=1}^{n}[(x-a_i)^2 + (y-b_i)^2] - nz^2 = 0.$$

Substituting

$$z^2 = \frac{1}{n} \cdot \sum_{j=1}^{n}[(x-a_j)^2 + (y-b_j)^2]$$

into the first two equations, we get the following system:

$$\sum_{i=1}^{n}(x-a_i)\{n[(x-a_i)^2 + (y-b_i)^2] - \sum_{j=1}^{n}[(x-a_j)^2 + (y-b_j)^2]\} = 0,$$
$$\sum_{i=1}^{n}(y-b_i)\{n[(x-a_i)^2 + (y-b_i)^2] - \sum_{j=1}^{n}[(x-a_j)^2 + (y-b_j)^2]\} = 0.$$

Apparently, this is a nonlinear system; yet, having performed all calculations, we finally obtain a linear system with the following unique solution:

$$x_0 = \frac{1}{2} \cdot \frac{B \cdot E - C \cdot D}{B^2 - A \cdot C} \tag{1}$$

$$y_0 = \frac{1}{2} \cdot \frac{B \cdot D - A \cdot E}{B^2 - A \cdot C} \tag{2}$$

where

$$A = \sum_{i=1}^{n}\sum_{j=1}^{n}(a_i - a_j)^2 = 2 \cdot \sum_{i=1}^{n-1}\sum_{j=i+1}^{n}(a_i - a_j)^2,$$

$$B = \sum_{i=1}^{n}\sum_{j=1}^{n}(a_i - a_j)(b_i - b_j) = 2 \cdot \sum_{i=1}^{n-1}\sum_{j=i+1}^{n}(a_i - a_j)(b_i - b_j),$$

$$C = \sum_{i=1}^{n}\sum_{j=1}^{n}(b_i - b_j)^2 = 2 \cdot \sum_{i=1}^{n-1}\sum_{j=i+1}^{n}(b_i - b_j)^2,$$

$$D = \sum_{i=1}^{n}\sum_{j=1}^{n}(a_i - a_j)(a_i^2 - a_j^2 + b_i^2 - b_j^2) = 2 \cdot \sum_{i=1}^{n-1}\sum_{j=i+1}^{n}(a_i - a_j)(a_i^2 - a_j^2 + b_i^2 - b_j^2),$$

$$E = \sum_{i=1}^{n}\sum_{j=1}^{n}(b_i - b_j)(a_i^2 - a_j^2 + b_i^2 - b_j^2) = 2 \cdot \sum_{i=1}^{n-1}\sum_{j=i+1}^{n}(b_i - b_j)(a_i^2 - a_j^2 + b_i^2 - b_j^2),$$

and the radius of the circle is:

$$z_0 = \sqrt{\frac{1}{n} \cdot \sum_{i=1}^{n}[(x_0 - a_i)^2 + (y_0 - b_i)^2]}.$$

## 3. Remarks and consequences

The fit model using the power of a point can be successfully applied also to conical (elliptic, hyperbolic or parabolic) fittings for finite planar point sets. Moreover, this model works very well in $\mathbb{R}^3$ to approximate a finite non coplanar point set by a spherical surface.

We'll end by mentioning the following obvious consequences:

**Corollary 1.** The points $P_i(a_i, b_i) \in \mathbb{R}^2, i = 1, \ldots, n$, where $2 \leq n < \infty$ are collinear if and only if:

$$\{\sum_{i=1}^{n}\sum_{j=1}^{n}(a_i - a_j)(b_i - b_j)\}^2 = \{\sum_{i=1}^{n}\sum_{j=1}^{n}(a_i - a_j)^2\} \cdot \{\sum_{i=1}^{n}\sum_{j=1}^{n}(b_i - b_j)^2\}.$$

The condition can be used for checking the (non)collinearity in problems related to certain geometrical data structures, such as the triangulations or the Voronoi diagrams [5].

**Corollary 2.** The points $P_i(a_i, b_i) \in \mathbb{R}^2, i = 1, \cdots, n$, where $3 \leq n < \infty$ are concyclic if and only if:

$$n \cdot \sum_{i=1}^{n}[(x_0 - a_i)^2 + (y_0 - b_i)^2]^2 = \{\sum_{i=1}^{n}[(x_0 - a_i)^2 + (y_0 - b_i)^2]\}^2$$

where $x_0, y_0 \in \mathbb{R}$ are given by (1) and (2), respectively.

We believe the above condition could be exploited in computational geometry as well.

## References

[1] I. Bass, *General equations of gear cutting tool calculations; Gear Technology*, The Journal of Gear Manufacturing, 2 (1985), pp. 20–23.

[2] G. Cimoca, *Asupra aproximării prin arce de cerc a mulţimilor finite de puncte din plan*, Al II-lea Simpozion de informatică şi conducere, vol. II, C.T.C.E. Cluj-Napoca, 1977, pp. 104–106.

[3] W.S. Dorn, D.D. McCracken, *Numerical Methods with FORTRAN IV Case Studies*, Wiley, New York, 1972.

[4] W. Mendenhall, T. Sincich, *A Second Course in Statistics: Regression Analysis*, Prentice Hall, Upper Saddle River, New York, 1996.

[5] F.P. Preparata, M.I. Shamos, *Computational Geometry: An Introduction*, Springer Verlag, New York, Heidelberg, 1985.

[6] I.A. Stoica, G. Cimoca, *Interferenţa de sarcină a angrenajelor (Load Interference of Gears)*, Editura Dacia, Cluj-Napoca, 1978.

S.C. SIMBOLIC, str. Bacău nr. 3, 3400 Cluj-Napoca, ROMÂNIA, tel: +40-64-431.333, fax: +40-64-199.895, e-mail: ghcimoca@symbolic.com

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLVI**, Number 2, 2001

# A REINFORCEMENT LEARNING INTELLIGENT AGENT

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 his performance. In this paper we design an Intelligent Agent who learns (by reinforcement) to play the "Game 21", a simplified version of the game "Black -Jack". The algorithm used for training the agent is the **SARSA** algorithm.

**Keywords**: Artificial Intelligence, Reinforcement Learning, Intelligent Agents.

## 1. Intelligent agents

An *agent* [1] is anything that can be viewed as **perceiving** its environment through *sensors* and **acting** upon that environment through *actions*. One of the task of an agent is to assist the user, achieving tasks in his place, or teaching the user what he should do. An *agent* is characterized by:

- the *architecture part*, or the agent's behavior - the action performed after any given sequence of percepts;
- the *program part*, or the agent's built-in part - the internal functionality of the agent.

The aim of Artificial Intelligence is to design the agent program: a function that implements the agent mapping from percepts to actions.

So, an artificial intelligent agent should be endowed with an *initial (built-in) knowledge* and with the capability of *learning*. The learning capability ensures the agent's *autonomy* - the capability of deducing his behavior from its own experience.

An intelligent agent has a *utility function*, which measure the performance of the agent's actions. The utility is a function that associates a real number to an agent's state, as a measure of the state's degree of success (a state preferred by the agent in comparison with others has a bigger utility function).

---

2000 *Mathematics Subject Classification.* 68T05.

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

As a conclusion, the *intelligence* of an agent is included in his program part. At a given moment, the agent will choose the best way of action, as he was programmed to do it. In situations in which the program has incomplete information (knowledge) about the environment in which the agent acts and lives, *learning* is the only way for the agent to acquire the knowledge he needs in order to achieve his task.

So, an important task is to design the program part of an intelligent agent, and even more, to implement the capability of learning.

## 2. Reinforcement Learning

*Reinforcement Learning* (RL) [3] is an approach to machine intelligence that combines two disciplines to solve successfully problems that neither discipline can address individually: the fields of dynamic programming and supervised learning. In RL, the computer is simply given a goal to achieve. The computer then learns how to achieve that goal by trial-and-error interactions with its environment.

A reinforcement learning problem has three fundamental parts [3]:

- *the environment* - represented by "states". Every RL system learns a mapping from situations to actions by trial-and-error interactions with a dynamic environment. This environment must at least be partially observable by the reinforcement learning system;
- *the reinforcement function* - the "goal" of the RL system is defined using the concept of a reinforcement function, which is the exact function of future reinforcements the agent seeks to maximize. In other words, there exists a mapping from state/action pairs to reinforcements; after performing an action in a given state the RL agent will receive some reinforcement (reward) in the form of a scalar value. The RL agent learns to perform actions that will maximize the sum of the reinforcements received when starting from some initial state and proceeding to a terminal state.
- *the value (utility) function* - explains how the agent learns to choose "good" actions, or even how we might measure the utility of an action. Two terms were defined: a policy determines which action should be performed in each state; a policy is a mapping from states to actions. The value of a state is defined as the sum of the reinforcements received when starting in that state and following some fixed policy to a terminal state. The value (utility) function would therefore be the mapping from states to actions that maximizes the sum of the reinforcements when starting in an arbitrary state and performing actions until a terminal state is reached.

In a reinforcement learning problem, the agent receives a feedback, known as *reward* or *reinforcement*; the reward is received at the end, in a terminal state, or

in any other state, where the agent has correct information about what he did well or wrong.

2.1. **Q-learning.** *Q-learning* [3] is another extension to traditional dynamic programming (value iteration) and solves the problem of the *non-deterministic* Markov decision processes, in which a probability distribution function defines a set of potential successor states for a given action in a given state.

Rather then finding a mapping from states to state values, Q-learning finds a mapping from state/action pairs to values (called *Q-values*). Instead of having an associated value function, Q-learning makes use of the Q-function. In each state, there is a Q-value associated with each action. The definition of a Q-value is the sum of the (possibly discounted) reinforcements received when performing the associated action and then following the given policy thereafter. An *optimal Q-value* is the sum of the reinforcements received when performing the associated action and then following the optimal policy thereafter.

If $Q(a, i)$ denotes the value of doing the action $a$ in state $i$, $R(i)$ denotes the reward received in state $i$ and $M_{ij}^{a}$ denotes the probability of reaching state $j$ when action $a$ is taken in state $i$, the Bellman equation for Q-learning (which represents the constraint equation that must hold at equilibrium when the Q-values are correct) is the following [1]:

$$Q(a, i) = R(i) + \sum_{j} \max_{a'} Q(a', j) \tag{1}$$

or, equivalently

$$Q(a, i) = R(i) + \gamma \cdot \max_{a'} Q(a', j) \tag{2}$$

The temporal-difference approach requires no model of the environment, so the update equation for TD Q-learning is:

$$Q(a, i) = Q(a, i) + \alpha(R(i) + \max_{a'} Q(a', j) - Q(a, i)) \tag{3}$$

which is calculated after each transition from state $i$ to state $j$, and $\alpha \in [0, 1]$ is called the *learning rate*.

2.2. **Selection mechanisms.** In this subsection we present two of the simplest action-selection mechanisms (policies).

*The Greedy and $\epsilon$-Greedy policy.* One of the simplest action-selection mechanism is the *Greedy* mechanism and its extension $\epsilon$ *-Greedy.* This mechanism operates this way:

- chooses the action with the maximum value Q(s,a), with the probability *1* - $\epsilon$, if it exists;
- otherwise, chooses a random action.

The $\epsilon$-Greedy methods are better than Greedy methods, because they allow exploration and continue to improve their chances for recognizing the optimal actions. *The SoftMax policy.* In a SoftMax policy, at the t-th game, action $a$ will be chosen with the probability:

$$\frac{e^{\frac{Q_t(a)}{\tau}}}{\sum_{b=1}^{n} e^{\frac{Q_t(b)}{\tau}}} \tag{4}$$

where $\tau$ is a positive parameter called *temperature*. The high temperatures make almost all actions to be equal-probable.

Low temperatures make a bigger difference between the selection probabilities of actions. At the limit, when $\tau \to 0$, the SoftMax action selections behave like Greedy action selections.

2.3. **The SARSA Algorithm.** SARSA is a reinforcement Q-learning algorithm, which combines the advantages of Temporal difference learning and Monte Carlo learning methods.

From the TD methods, the algorithm takes the advantage of learning at each step, instead of waiting the end of an episode. From the Monte Carlo methods, SARSA takes the advantage of going back and using the rewards obtained in each state in order to update the values of the previous action-state pairs and the capacity of functioning without the model of the environment in which the learning takes place [4].

The idea of the SARSA algorithm [2]is to apply the Temporal Difference methods to the state-action pairs, in comparison with the classical methods, where this methods are applied only to the states. So, the update equation for the action-state pairs will be :

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{5}$$

This update is done after every transition from a nonterminal state $s_t$. If $s_{t+1}$ is terminal, then $Q(s_{t+1}, a_{t+1})$ is defined as 0. This rule uses every element of the quintuple of events, $s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1}$, that make up a transition from one state-action pair to the next. This quintuple gives rise to the name SARSA for the algorithm.

The convergence properties of the SARSA algorithm depends on the nature of the policy's dependence on Q (one could use $\epsilon$-Greedy or SoftMax policies). SARSA converges with probability 1 to an optimal policy and action-value function as long as all state-action pairs are visited an infinite number of times and the policy converges in the limit to the Greedy policy.

The general form of the SARSA algorithm is given in Figure 1.

Initialize $Q(s, a)$ arbitrarily
Repeat(for each episode)
Initialize $s$
Choose $a$ from $s$ using policy derived from $Q$($\epsilon$-Greedy, SoftMax)
Repeat(for each step of the episode):
Take action $a$, observe $r$, $s'$
Choose $a'$ from $s'$ using policy derived from $Q$($\epsilon$-Greedy, SoftMax)
$Q(s, a) = Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)]$
s $\rightarrow$ s';
a $\rightarrow$ a';
until $s$ is terminal

FIGURE 1. The SARSA Algorithm

## 3. The application "Agent 21"

3.1. **General presentation.** In this section we describe an Intelligent Agent which learns by reiforcement to play the "Game 21". We propose a game with two players: a dealer (which shares the cards) and an agent, in fact the computer (which uses the SARSA algorithm for learning the game's rules and which plays against the dealer). The agent tries to win, obtaining a score (the sum of the cards' values) less or equal than 21, but greater than the dealer's score.

The probabilistic nature of this game make it an interesting problem for testing the learning algorithms, even if the learning of a good playing strategy is not obvious.

Learning from a teacher (the supervised learning) is not useful in this game, as the output data for a given state are unknown. The agent has to explore different actions and to develop a strategy by selecting and retaining the actions that maximize his performance.

The purpose of this application is to study the behavior of an agent, which initially doesn't know neither the rules nor the game's goal, and to follow the way for assimilating this goals.

After learning, the agent (the computer) wins about 54% of the games. This is due to the fact that the "Game 21" is a random game. The agent tries to manage as well as he can with the cards received from the dealer.

As an observation, even if the game has a strong probabilistic characteristic, after the training rounds the agent plays approximately identically or maybe even better than a human player (which has a random playing strategy).

3.2. **The application.** The application is written in Visual Basic 6.0 and has two parts:

- *Learning*, which trains the agent using the SARSA algorithm. Before starting the learning process, the user can select:
  - the number of episodes of the agent's training and the number of games per episode;
  - the rewards obtained by the agent when he wins or looses, the values for $\alpha$ and $\gamma$ (the step size and the reducing factor from the basis equation of the SARSA algorithm);
  - the selection mechanisms ($\epsilon$-Greedy or SoftMax). After the learning process, the user can see a graphic that contains the percentage of the games that the agent won and the values of the action-state pairs.
- *Game*, which simulates the game between the agent and the dealer. Before starting the game, the user can select the score at which the dealer stops, the strategy used by the agent. If the learning took place, the *current strategy* can be selected and the agent will use the experience from the training games, otherwise a *random strategy* can be selected and the agent will play randomly.

3.3. **The Agent's design.** The basis classes used for implementing the agent's behavior are the following:

- **CGame** - contains functions for starting the game, for calculating the points, calls the functions for design the cards on the game table and at the end of the game displays if a player wins or looses;
- **CPlayer** - contains the settings for the players: for the computer adds the current strategy. In this class are called the functions for designing the cards on the game table, for calculating the score. The main method of the class is *ComputerPlay*, which implements the game function for the computer;
  - the computer applies the selected strategy. If the agent has gone through all the training rounds, he will use his experience and will apply the strategy used in the training rounds ($\epsilon$-Greedy or Softmax).
- **CPlayers** - is the class for creating the players and setting the game strategy for the dealer and the computer;
- **CReinforcementLearning** - the main class of the agent which implements the SARSA algorithm. Here the learning function for the agent is

implemented and the estimation function is updated. The main method is *Learn*.

The learning strategy is very conservative: it stops if the score is greater then 11. Anyway, it is very interesting that the algorithm determines such a threshold without knowing the game's rules or it's goal (only from experience and from the obtained rewards). In fact, the SARSA agent learns to continue (to ask for one more card) only if he is sure that this will not make him to loose. The agent discovers from experience the double value of the ace, which determines the following strategy: continues if the score is lower than 11 or he has an ace; stops in all the other situations.

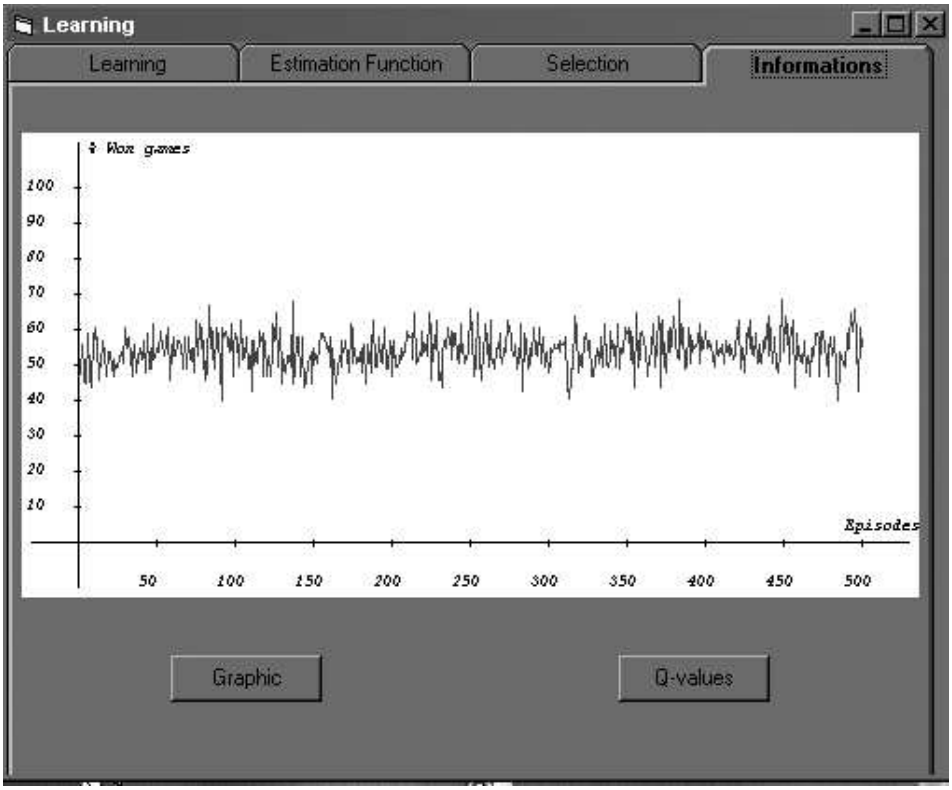


FIGURE 2

3.4. **Experiment.** Using the application "Agent 21" we accomplish the training of the agent for the following settings:

- $\alpha = 0.01$;
- $\gamma = 0.9$;
- the reward for winning the game = 1;
- the reward for loosing the game = -1;
- number of episodes = 500;
- number of games/episode = 100.

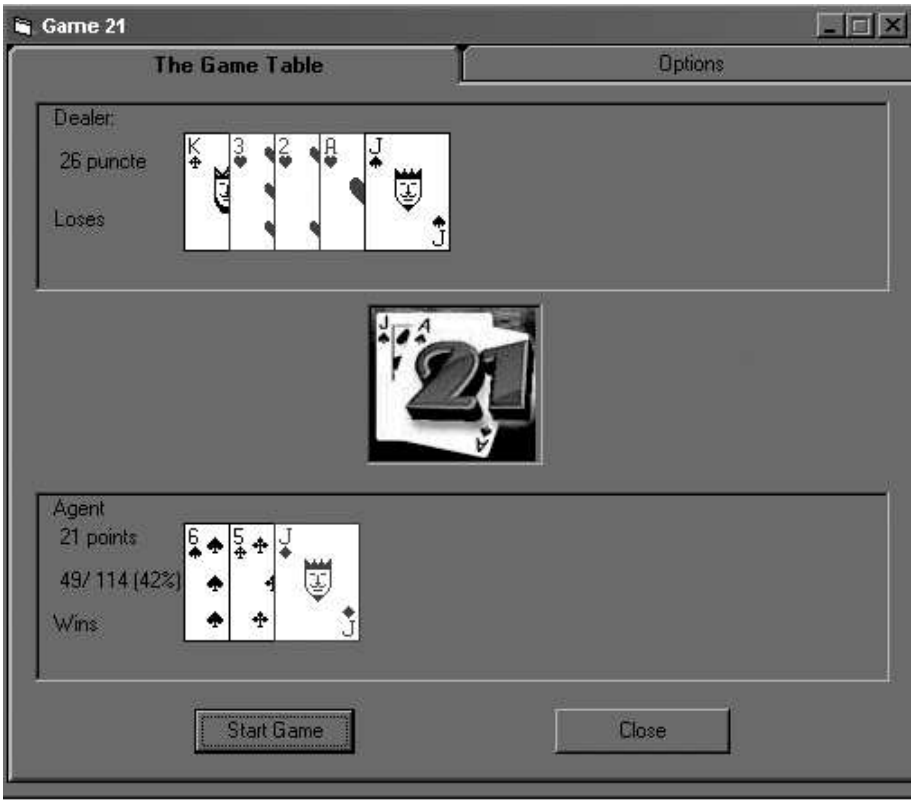


FIGURE 3. The game table with the agent's training

During the learning process, the agent wins 27188 games and loses 22812 games, that is a percentage of 54.376% of the won games. The graphic that contains the percentage of the games that the agent won during the learning is shown in Figure 2.

After the training took place, the agent uses his experience and wins about 50% of the the games (Figure 3).

Without training (with a random strategy), the agent won about 30% of the games (Figure 4).

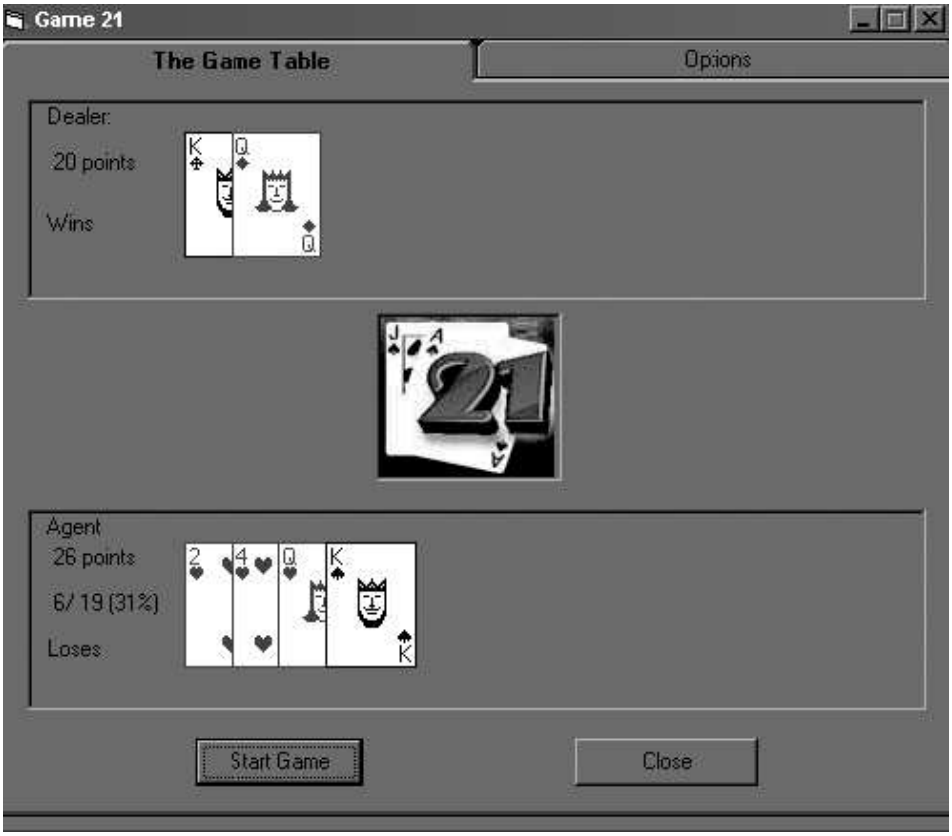


FIGURE 4. The game table without the agent's training

## 4. CONCLUSIONS

The agent implemented using the SARSA algorithm in the "Game 21" uses an optimal strategy. These allows him to win about 50% of the games, because of the strong random characteristic of the game. Knowing that, using a random strategy, the computer wins only about 30%, it's easy to notice the intelligence degree obtained by the agent after learning. However, if a human player and an agent, who assimilated the optimal strategy, play against the dealer, it can be observed that the number of the games won by the human player is approximately equal to the number of the games won by the agent.

Anyway, Reinforcement Learning represents an important way for improving the performance and the behavior of an Intelligent Agent.

## References

[1] S.J.Russell, P.Norvig: "Artificial intelligence. A modern approach", Prentice-Hall International, 1995

[2] R. Sutton, A. Barto: "Reinforcement Learning", The MIT Press, Cambridge, London, 1998

[3] M. Harmon, S. Harmon: "Reinforcement Learning — A Tutorial", Wright State University, www-anw.cs.umass.edu./ mharmon/rltutorial/frames.html, 2000

[4] A. Perez-Uribe, E. Sanchez: "Blackjack as a TestBed for Learning strategies in Neural Networks", Proceedings of the IEEE International Joint Conference on Neural Networks IJCNN'98

Babeş-Bolyai University, Department of Computer Science, Cluj-Napoca, Romania
*E-mail address*: gabis@cs.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLVI**, Number 2, 2001

# A NEW EVOLUTIONARY ADAPTIVE REPRESENTATION PARADIGM

D. DUMITRESCU, CRINA GROŞAN, MIHAI OLTEAN

Abstract. In this paper a new evolutionary paradigm is proposed. A technique called Adaptive Representation Evolutionary Algorithm (AREA) based on this paradigm is designed. AREA involves dynamic alphabets for encoding solutions. Proposed adaptive representation is more compact than binary representation. Mutation is the unique variation operator. Mutations are usually more aggressive when higher alphabets are used. Therefore the proposed encoding ensures an efficient exploration of the search space.

Numerical experiments seem to indicate that APA process better than the best multiobjective evolutionary algorithms.

An AREA technique is used for solving multiobjective optimization problems. The resulting algorithm is called Adaptive Pareto Algorithm (APA).

**Keywords**: Evolutionary Computation, Evolutionary Multiobjective Optimization, Pareto Set, Higher Alphabet Encoding, Adaptive Representation.

## 1. Introduction

In this paper we propose a new evolutionary paradigm. An algorithm based on this paradigm and using a powerful adaptive representation is designed. The algorithm called Adaptive Representation Evolutionary Algorithm (AREA). AREA technique operators are mutation and selection for survival.

Many multiobjective optimization techniques using evolutionary algorithms have been proposed in recent years. Strength Pareto Evolutionary Algorithm (SPEA, [9]), Pareto Archived Evolution Strategy (PAES, [4]), Pareto Envelope – based Selection Algorithm (PESA, [1]), Nondominated Sorting Genetic Algorithm (NSGA II, [3]) and SPEA II ([10]) are the best present-day Multiobjective Evolutionary Algorithms (MOEAs).

Multi-alphabet representation proposed in this paper induces a powerful diversity maintaining mechanism. For this reason AREA technique seems to be very suitable for evolutionary multiobjective optimization purposes. Considered adaptive encoding allows solutions in the final population realizing a realistic picture of Pareto frontier.

---

2000 *Mathematics Subject Classification.* 68T05.

1998 *CR Categories and Descriptors.* I.2.8 [**Computing Methodologies**]: Artificial Intelligence – *Problem Solving, Control Methods, and Search.*

Numerical experiments with APA technique include several test functions reputated as difficult ([2], [7]) and comparisons with the best MOEAs.

The paper is structured as follows: Section 2 is a short resume of the principal recent evolutionary techniques for multiobjective optimization. Section 3 describes the proposed algorithm. In Section 4 a comparison of the proposed approach with some very efficient multiobjective evolutionary techniques is realized.

## 2. Recent MOEAs

In the last years a number of evolutionary algorithms for multiobjective optimization have been proposed. Some of them will be shortly reviewed here.

2.1. **Strength Pareto Evolutionary Algorithm.** Zitzler and Thiele proposed an elitist evolutionary algorithm called Strength Pareto Evolutionary Algorithm (SPEA) ([9, 7]). The algorithm maintains an external population at every generation storing all nondominates solutions obtained so far. At each generation external population is mixed with the current population. All nondominated solutions in the mixed population are assigned fitness based on the number of solutions they dominate. Dominated solutions are assigned fitness worse than the worst fitness of any nondominates solutions. A deterministic clustering technique is used to ensure diversity among nondominates solutions.

**Pareto Archived Evolution Strategy**

Knowles and Corne [4] proposed a simple evolutionary algorithm called Pareto Archived Evolution Strategy (PAES). In PAES one parent generates by mutation one offspring. The offspring is compared with the parent. If the offspring dominates the parent, the offspring is accepted as the next parent and the iteration continues. If the parent dominates the offspring, the offspring is discarded and the new mutated solution (a new offspring) is generated. If the offspring and the parent do not dominate each other, a comparison set of previously nondominated individuals is used.

For maintaining population diversity along Pareto front, an archive of nondominated solutions is considered. A new generated offspring is compared with the archive to verify if it dominates any member of the archive. If yes, then the offspring enters the archive and is accepted as a new parent. The dominated solutions are eliminated from the archive. If the offspring does not dominate any member of the archive, both parent and offspring are checked for their nearness with the solution of the archive. If the offspring resides in the least crowded region in the parameter space among the members of the archive, it is accepted as a parent and a copy is added to the archive.

2.2. **Nondominated Sorting Genetic Algorithm.** Deb and his students [3] suggested a fast elitist Nondominated Sorting Genetic Algorithm (NSGA II). In NSGA II, for each solution $x$ the number of solutions that dominate solution $x$ is calculated. The set of solutions dominated by $x$ is also calculated. The first front (the current front) of the solutions that are nondominated is obtained.

Let us denote by $S_i$ the set of solutions that are dominated by the solution $x^i$. For each solution $x^i$ from the current front consider each solution $x^q$ from the set $S_i$. The number of solutions that dominates $x^q$is reduced by one. The solutions that remain nondominates after this reduction will form a separate list. This process continues using the newly identified front as the current front.

Let $P(0)$ be the initial population of size $N$. An offspring population $Q(t)$ of size $N$ is created from current population $P(t)$. Consider the combined population:

$R(t) = P(t) \quad \cup \quad Q(t)$.

Population $R(t)$ is ranked according nondomination. The fronts $F_1$, $F_2$, ... are obtained. New population $P(t+1)$ is formed by considering individuals from the fronts $F_1$, $F_2$, ..., until the population size exceeds $N$. Solutions of the last allowed front are ranked according to a crowded comparison relation.

NSGA II uses a parameter (called *crowding distance*) for density estimation for each individual. Crowded distance of a solution $x$ is the average side-length of the cube enclosing the point without including any other point in the population. Solutions of the last accepted front are ranked according to the crowded comparison distance.

NSGA II works as follows. Initially a random population, which is sorted based on the nondomination, is created. Each solution is assigned a fitness equal to its nondomination level (1 is the best level). Binary tournament selection, recombination and mutation are used to create an offspring population. A combined population is formed from the parent and offspring population. The population is sorted according to the nondomination relation. The new parent population is formed by adding the solutions from the first front and the followings until exceed the population size. Crowding comparison procedure is used during the population reduction phase and in the tournament selection for deciding the winner.

## 3. AREA technique

In this paper we propose a new evolutionary paradigm, The main idea is to allow each solution be encoded on a different alphabet. Moreover representation of a particular solution is not fixed. Representation is adaptive and may be changed during the search process as an effect of mutation operator, An adaptive representation evolutionary algorithm (AREA) based on the new paradigm is designed.

AREA technique proposed in this paper uses a fixed population. Each AREA individual (chromosome) consists of a pair ($x$, $B$), where $x$ is a string encoding object variables and $B$ specifies the alphabet used for encoding $x$.

$B$ is an integer number, $B \quad \geqslant 2$ and $x$ is a string of symbols from the alphabet $\{0, 1, \ldots, B\text{-}1\}$. If $B= 2$, the standard binary encoding is obtained. The alphabet over which $x$ is encoded may change during the search process.

Mutation is the unique variation operator. For mutation, a random number between 0 and 1 is uniformly generated for each position, including the last one, of the chromosom. Each position (gene) value is modified with a mutation probability ($p_m$).

Mutation can modify object variables as well as last position (fixing the representation alphabet). If the position giving $B$ is changed, then the object variables will be represented using symbols over the new alphabet, corresponding to the mutated value of $B$. When the changing gene belongs to the object variable sub-string ($x$ – part of the chromosome), the mutated gene is a symbol randomly chosen from the same alphabet.

## 4. APA method

In this section a new MOEA technique called Adaptive Pareto Algorithm (APA) is proposed. APA relies on the AREA method previously described.

AREA uses a unique population. No external or intermediary population is needed.

Initial population is randomly generated. Each individual is selected for mutation, which is the unique variation operator. The offspring and parent are compared. Dominance relation guides the survival.

If the offspring dominates the parent then the offspring enters the new population and the parent is removed. If the parent dominates the offspring obtained in $k$successive mutations then another alphabet is chosen and the parent is represented in symbols over this alphabet. In this case only representation is changed and the encoded solution does not change. Adaptive representation mechanism and the survival strategy is generates an effective and efficient diversity preserving mechanism.

**APA algorithm**

Proposed APA algorithm may be outlined as follows:

**APA** ALGORITHM:

**begin**
**Set** $t = 0$;
Random initializes chromosome population $P$ $(0)$;
Set to zero the number of harmful mutations for each individual in P(0);
**while** ($t$ ¡ number of generations) **do**
**begin**
**for** k = 1 **to** PopSize **do**
Mutate the k$^{th}$ chromosome from $P$ $(t)$. An offspring is obtained.
**If** the offspring dominates the parent then the parent is removed and the offspring is added to $P(t+1)$;
**else begin**
Increase the number of harmful mutations for current individual;
**If** the number of harmful mutations = MAX_HARMFUL_MUTATIONS
**then begin**
Change the individual representation;
Set to zero the number of harmful mutations for the current individual;
**endif**
**Add** individual to $P(t+1)$;

**endif**
**endfor**;
**Set** $t = t + 1$;
**endwhile;**
**end.**

Despite its simplicity APA is able to generate a population converging towards Pareto optimal set. Moreover, the diversity of the population is automatically maintained without any specialized mechanism.

Proposed APA algorithm realizes a realistic picture of Pareto optimal set. Numerical experiments emphasizes that for considered problems, APA technique is more effective then best present-day MOEAs. Moreover, APA's complexity is a reduced one with respect to the MOEAs techniques considered for comparison.

## 5. Comparison of Several Evolutionary Multiobjective Algorithms

In this section complexity of the proposed APA technique is compared with the complexity of several evolutionary multiobjective optimization algorithms (SPEA, PAES and NSGA II).

Let us denote by $m$ the number of objectives and by $N$ the population size.

SPEA uses an internal and an external population. The fitness is assigned differently to the individuals from these populations. A deterministic clustering technique is used to reduce external population size the population diversity. The complexity of this algorithm implementation is $mN^2$.

PAES uses a single parent, which generate an offspring. An archive, which maintains the nondominated solutions, is created. Let be $a$ the archive size. The worst case complexity for the PAES is $amN$. Since the archive size is usually proportional to the population size $N$,the overall complexity of the algorithm is $mN^2$.

NSGA II computes for each individual $x$ the number of solutions that dominates it and the number of solution, which $x$ dominates. NSGA II uses for this $N^2$ computations. Identifying the fronts requires (in the worst case) $N^2$ computations. The overall complexity is $(mN^2 + N^2)$ or $N^2$. So, the complexity may increase from $N$ to $N^2$ (the worst case). Computation complexity for density estimation is $mN\ \log N$. For sorting the combined population $2N\log(2N)$ computations are necessary. Overall NSGA II complexity is thus $mN^2$.

APA uses a unique fixed size population. Each individual is considered for mutation. This requires $mN$ operations. The algorithm does not use a superposed mechanism for diversity maintaining. Overall complexity of SMEA algorithm is thus $O(mN)$.

## 6. Numerical experiments

In this section we compare the performance of APA algorithm with the performances of SPEA, NSGA II, PAES.

For this purpose by using six test functions introduced by Deb, Zitzler and Thiele [7] are used.

6.1. **Test functions.** Each test function considered in this section is built by using three functions $f_1$, $g$, $h$. Biobjective function $T$ considered here is

$T(x) = (f_1(x), f_2(x))$.

The optimization problem is:

$$\begin{cases} Minimize\, T(x),\ where\, f_2(x) = g(x_2, \ldots, x_m)h(f_1(x_1), g(x_2, \ldots x_m), \\ x = (x_1, \ldots, x_m) \end{cases}$$

The five test functions used in this paper for comparison are:

Test function $T_1$ is defined using the following functions:

$$\begin{aligned} &f_1(x_1) = x_1, \\ &g(x_2, \ldots .x_m) = 1 + 9 \cdot \textstyle\sum_{i=2}^{m} x_i/(m-1), \\ &h(f_1, g) = 1 - \sqrt{f_1/g}, \end{aligned}$$

where $m$= 30 and $x_i \in$ [0,1] $i$ = 1,2,...,$m$.

Pareto optimal front for the problem $T_1$ is convex and is characterized by the equation

$$g(x) = 1.$$

Test function $T_2$ is defined by considering the following functions:

$$\begin{aligned} &f_1(x_1) = x_1 \\ &g(x_2, \ldots .x_m) = 1 + 9 \cdot \textstyle\sum_{i=2}^{m} x_i/(m-1) \\ &h(f_1, g) = 1 - (f_1/g)^2 \end{aligned}$$

where $m$ = 30 and $x_i \in$ [0,1], $i$ = 1,2,...,$m$.

Pareto optimal front is characterized by the equation

$$g(x)=1.$$

$T_2$ is the nonconvex counterpart to $T_1$.

Pareto optimal set corresponding to the Test function $T_3$ presents a discreten feature. Pareto optimal front consists of several noncontiguous convex parts. The involved functions are:

$$\begin{aligned} &f_1(x_1) = x_1 \\ &g(x_2, \ldots .x_m) = 1 + 9 \cdot \textstyle\sum_{i=2}^{m} x_i/(m-1) \\ &h(f_1, g) = 1 - \sqrt{f_1/g} - (f_1/g)\sin(10\pi f_1) \end{aligned}$$

where $m$= 30 and $x_i \in$ [0,1], $i$ = 1,2,... $m$.

Pareto optimal front is characterized by the equation

$$g(x) = 1.$$

The introduction of the function *sin* in the expression of function $h$ causes discontinuity in the Pareto optimal front. However, there is no discontinuity in the parameter space.

The test function $T_4$ contains $21^9$ local Pareto optimal fronts and, therefore, it tests the EA ability to deal with multimodality. The involved functions are defined by:

$$\begin{aligned} &f_1(x_1) = x_1 \\ &g(x_2, \ldots .x_m) = 1 + 10(m-1) + \textstyle\sum_{i=2}^{m} (x_i^2 - 10\cos(4\pi x_i)) \\ &h(f_1, g) = 1 - \sqrt{f_1/g} \end{aligned}$$

where $m = 10$, $x_1 \in$ [0,1] and $x_2, \ldots, x_m \in$ [-5,5].

Global Pareto optimal front is characterized by the equation

$$g(x) = 1.$$

The best local Pareto optimal front is described by the equation

$$g(x) = 1.25.$$

Note that not all local Pareto optimal sets are distinguishable in the objective space.

$$\begin{aligned} &f_1(x_1) = 1 - \exp(-4x_1)\sin^6(6\pi x_1) \\ &g(x_2, \ldots .x_m) = 1 + 9 \cdot \left(\textstyle\sum_{i=2}^{m} x_i/(m-1)\right)^{0.25} \\ &h(f_1, g) = 1 - (f_1/g)^2 \end{aligned}$$

The test function $T_5$ includes two difficulties caused by the nonuniformity of the search space. First, the Pareto optimal solutions are nonuniformly distributed along the global Pareto optimal front (the front is biased for solutions for which $f_1(x)$ is neat one). Second, the density of the solutions is lowest near the Pareto optimal front and highest away from the front.

This test function is defined by using:
where $m = 10$, $x_i \in$ [0,1], $i = 1,2,\ldots m$.

The Pareto optimal front is characterized by the equation

$$g(x) = 1$$

and is nonconvex.

6.1.1. *Numerical comparisons.* Several numerical experiments were performed with APA. According to these experiments APA gives a good approximation of the Pareto front for all considered test functions.

For both test functions $T_1$ and $T_2$, the differences between the four considered algorithms are very small (see Figure 1 and Figure 2).

The difference between APA and the other algorithms is significant for test function $T4$. APA gives the best arrangement on the front. Good solution distribution is obtained also by NSGA II and SPEA. Solutions distribution realized by NSGA II and SPEA are close to Pareto front. Moreover solution only distribution supplied by APA is covers the real front.

For test function $T_5$, APA also gives the best solution arrangement on the Pareto front. PAES also gives distribution.

In these comparisons 25.000.000 function evaluations have been considered for each algorithm. This ensures a realistic comparison of the algorithm outputs.

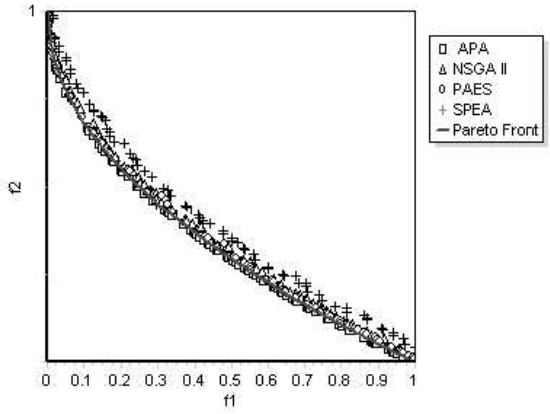


FIGURE 1. Results for test function $T_1$. Pareto optimal front is convex

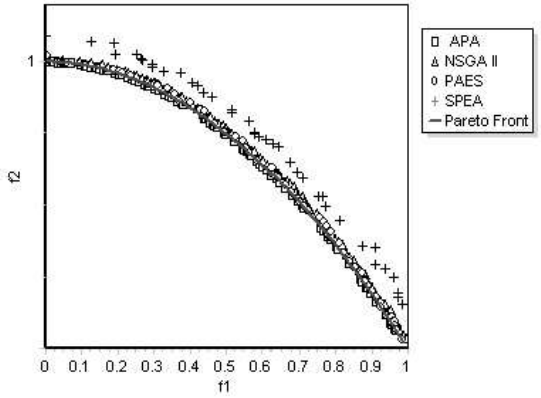


FIGURE 2. Results for test function $T2$. Pareto optimal front is nonconvex

**Conclusions**

In this paper a new evolutionary paradigm is proposed. An evolutionary algorithm (AREA) based on the new paradigm is designed.

A new evolutionary algorithm (called APA) for multiobjective optimization is also proposed. AREA uses a new, dynamic solution representation.

APA technique is compared with four well-known evolutionary multiobjective optimization algorithms. The results show that APA performs better than considered algorithms.

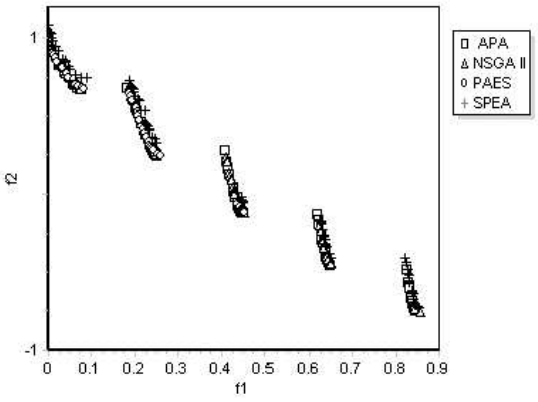


FIGURE 3. Results for test function $T3$. All considered algorithms give a good approximation of the Pareto front

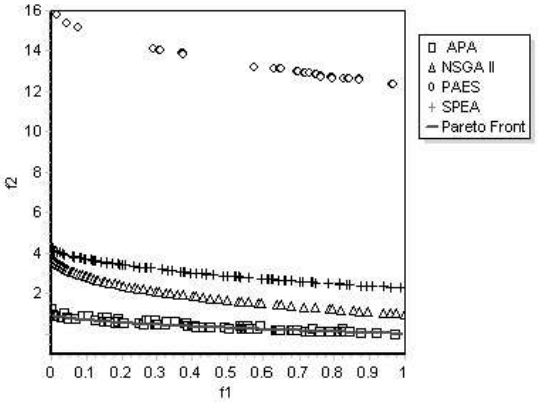


FIGURE 4. Results for test function $T_4$. For test function $T_4$, APA gives the best arrangement on the Pareto front. NSGA II and SPEA converge toward global Pareto front. PAES did not converge to the global Pareto front

## REFERENCES

[1] Deb, K., Multi-objective genetic algorithms: Problem difficulties and construction of test functions. *Evolutionary Computation*, 7(3), (1999), pp. 205-230.

[2] Deb, K., S. Agrawal, Amrit Pratap and T. Meyarivan, A fast elitist non – dominated sorting genetic algorithm for multi-objective optimization: NSGA II. In M. S. et al. (Ed), *Parallel Problem Solving From Nature – PPSN VI*, Berlin, (2000), pp. 849 – 858. Springer.

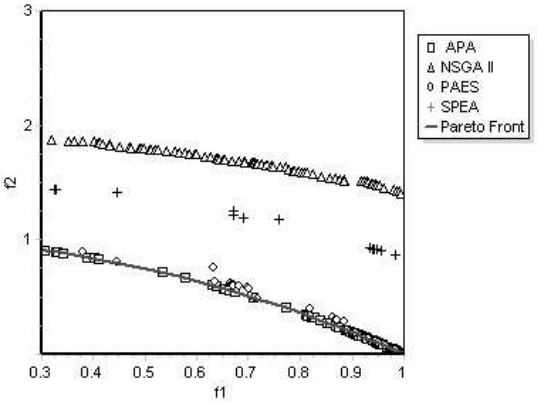


FIGURE 5. Results for test function $T_5$

[3] Dumitrescu, D., Groşan, C., Oltean, M., Simple Multiobjective Evolutionary Algorithm, Seminars on Computer Science, Faculty of Mathematics and Computer Science, Babeş-Bolyai University of Cluj-Napoca, 2001, pp. 3-12.

[4] Knowles, J. D. and Corne, D. W., The Pareto archived evolution strategy: A new baseline algorithm for Pareto multiobjective optimization. In *Congress on Evolutionary Computation (CEC 99)*, Volume 1, Piscataway , NJ, (1999), pp. 98 – 105. IEEE Press.

[5] Rudolph, G., On a multi – objective evolutionary algorithm and its convergence to the Pareto set. Technical Report No. CI – 17/98, Department of Computer Science/XI, university of Dortmund, (1998).

[6] Zitzler, E., Evolutionary algorithms for multiobjective optimization: Methods and Applications. Ph. D. thesis, Swiss Federal Institute of Technology (ETH) Zurich, Switzerland. TIK – Schriftenreiche Nr. 30, Diss ETH No. 13398, (1999), Shaker Verlag, Aachen, Germany.

[7] Zitzler, E., Deb, K. and Thiele, L., Comparison of multiobjective evolutionary algorithms: empirical results. Technical report 70, Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH), (1999), Zurich.

[8] Zitzler, E. and Thiele, L., Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto approach. *IEEE Transaction on Evolutionary Computation 3*(4), (1999), pp. 257 – 271.

[9] Zitzler, E. and Thiele, L., An evolutionary algorithm for multiobjective optimization: The strength Pareto approach. Technical report 43, Computer engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH), (1999), Zurich.

"BABEŞ-BOLYAI" UNIVERSITY, FACULTY OF MATHEMATICS AND COMPUTER SCIENCE, 1 M. KOGĂLNICEANU STREET, RO-3400 CLUJ-NAPOCA, ROMANIA

*E-mail address*: ddumitr|cgrosan|moltean@cs.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLVI**, Number 2, 2001

# AN EFFICIENCY COMPARISON OF DIFFERENT JAVA TECHNOLOGIES

FLORIAN MIRCEA BOIAN

Abstract. Java and related technologies are very used for distributed applications today. In this moment, there are many Java technologies using the client-server paradigm. Among these, the following are the most important: Common Gateway Interface (CGI) [4,7], Servlets/JSP [1,10,12], JavaSpaces [1,5,6] and Enterprise Java Beans (EJB) [1,2,8]. To choose one of these for solving a problem (to implement a distributed application), the performances, rapid development and robustness are important criteria. In this paper, we present a run-time performance comparison between these technologies.

## 1. Introduction

Platform independence is an important argument for using Java technologies instead others, (for example Microsoft's) distributed technologies. Each of the CGI, Servlet, JavaSpaces and EJB technologies has specific characteristics and implementation difficulties. Theoretically, they are, equivalent: every application implementation using a technology, from one of above, can be (theoretically) implemented using any of the other three technologies.

Of course, from practical point of view there are significant differences. The design and coding effort is relatively reduced for CGI, a little bit more significant for Servlet, a medium one for JavaSpaces and quite impressive for EJB. Taking into account the robustness, security and reliability, we have a reverse order: in the top is EJB; on last position is CGI. It's a difficult, and almost an impossible task, to analyse an compare these technologies in a global and unitary manner. Our opinion is that one has to solve and implement a (some) application(s) using each of the above technologies. Then, make a comparison and behavior analysis of the implementations vis--vis to a (some) criterion(s).

In the present paper, we solve a unique problem: a counter. Four implementations were made: CGI, Servlet, JavaSpaces and EJB. For these implementations, we analyse a single criterion: run-time aspect, both in the server part and client part.

In the next section, the test problem is presented. In the following four sections, the specific architecture and particularities for each implementation are presented.

---

2000 *Mathematics Subject Classification.* 68M14.

1998 *CR Categories and Descriptors.* C.2.4 [**Computer Systems Organizations**]: Computer-Communication Networks – *Distributed Systems.*

In the last section, some numerical results, comparison between these implementations and some conclusions are presented.

## 2. The experiment problem: a counter

Most programs for Internet applications are designed using the client / server paradigms and their extensions [3,9]. For our experiments, we use some counter implementations. Particularly, in Internet there are many counters. For example, many Web pages have counters for measuring the total number of accesses to them. The systems for voting popularity pages are based also to counters.

For our purpose, a counter is a pair of the following form:

```
(name, value),
```

where `name` is a string – the name of the counter – and `value` is the value of the counter – a positive integer.

There are two operations, defined as follows (like Java method prototypes):

```
void counterInit(String name, int initValue);

int counterAccess(String name);
```

`counterInit` creates the counter name and sets its initial value to `intValue`. `counterAccess` increments, for the counter name, the current value and returns it. Usually, the arithmetic overflow is ignored.

The distributed counter architecture is presented in Figure 1.

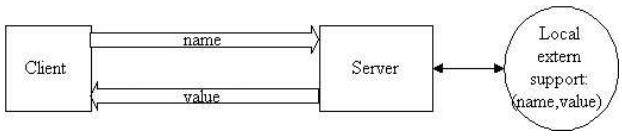


Figure 1. Distributed counter

We adapt the counter implementation so that we obtain the run-times per operation, for client and for server. In the first step, the client invokes the server to create the counter. For our goals, the `counterAccess` returns the run-time – in milliseconds.

The algorithm for the main step, in both the client and server, is presented in Table 1.

The experiment, for each technology, uses two hosts, the same for all technologies: one for the client – only one – and the other for the server. The above algorithms run, without interrupt and without sleeps, waits and so on, for few hours. The connection between hosts was made directly, without any gateways, proxy-s or other related entities. The hosts were run only necessaries services for the experiment.

From time to time, the client saves on a local file, the following information (times are in milliseconds):

- average server time for a `counterAccess`;

TABLE 1

| Client | Server |
|---|---|
| `for ( ; ; ) { // loop forever`<br><br>`  ti = theCurrentTime`<br>`  connect to server`<br><br>`  ts = counterAccess`<br><br>`  close connection`<br>`  tf = theCurrentTime`<br><br>`  use ti, ts, tf for statistics`<br>`  save, from time to time`<br>`  the intermediate statistics`<br>`  on a local file`<br><br>`}` | `for ( ; ; ) { // loop forever`<br><br>`  wait for connection`<br>`  ti = theCurrentTime`<br><br>`  open access from counter`<br>`  access the counter from the`<br>`  external support`<br>`  increment the value`<br>`  save the counter to this`<br>`  external support`<br>`  close access to counter`<br><br>`  tf = theCurrentTime`<br>`  return (tf - ti)`<br><br>`}` |

- average client time for a `counterAccess`;
- maximum server time for a `counterAccess`;
- maximum client time for a `counterAccess`;
- minimum server time for a `counterAccess`;
- minimum client time for a `counterAccess`;
- total server time spend for all `counterAccess`;
- total client time spend for all `counterAccess`;
- total connected time for client: the above time plus the time spent for computed statistics and save intermediate results;
- total number of connections.

The code sources used in the experiment can be accessed at the home page of the author: `http://www.cs.ubbcluj.ro/~florin`. For a uniform approach, and independently of technology, the clients are Java standalone applications, of course, very similar.

The client runs under Windows 2000 Pro, using an AMD K7 (Athlon) at 1GHz, with 512RAM. The server runs under Linux RedHat 7.2, also an AMD K7 (Athlon) at 1GHz, with 512RAM.

## 3. CGI: features and specifics

The Common Gateway Interface (CGI) [4,7] is a standard for interfacing external applications with information servers, such as HTTP or Web servers. A plain HTML document that the Web daemon retrieves is static, which means it exists in a constant state: a text file that doesn't change. A CGI program, on the other hand, is executed in real-time, so that it can output dynamic information. Since

a CGI program is executable, it is basically the equivalent of letting the world run a program on your system, which isn't the safest thing to do. Therefore, there are some security precautions that need to be implemented when it comes to using CGI programs. Probably the one that will affect the typical Web user the most is the fact that CGI programs need to reside in a special directory, so that the Web server knows to execute the program rather than just display it to the browser. This directory is usually under direct control of the webmaster, prohibiting the average user from creating CGI programs. Interested reader can find details about CGI in [4,7].

For our experiment, the CGI architecture is presented in Figure 2.

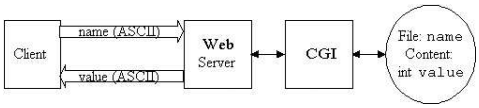


FIGURE 2. CGI architecture

The java standalone client sends to the CGI, using `URLConnection`, the name of the file using the POST method.

The CGI program was developed in ANSI C. For each connection, it opens the (binary) file name having only 4 bytes, in fact an int. Then it reads the value, increments it, rewrites it again and then close the file.

The response is a Content-type: `text/plain` and is the server time, in ASCII form.

## 4. Servlet: features and specifics

Servlets are the answer of Java technology's to CGI programming [1,10,11,12]. They are programs that run on a Web server and build Web pages. Java servlets are more efficient, easier to use, more powerful, more portable, and cheaper than traditional CGI and than many alternative CGI-like technologies.

As known, for using servlets, a servlet container is necessary. We use the Tomcat container - today a reference servlet container (and free distributable and license free).

For our purposes, only a simple architecture is necessary, so we use only the web facilities offered by the Tomcat 4.x version. Of course, for an important and consistent application, the interfce between Tomcat and Apache Web server must be use.

For our experiment, the Servlet architecture is presented in Figure 3.

The java standalone client sends to the servlet, using `URLConnection`, the name of the file using the POST.

The servlet implements the doPost method. Using the `HttpServletRequest` parameter, the name of the counter is obtained, reading an object String. For each connection, as in the case of CGI, does it open the (binary) file name having only 4 bytes, in fact an `int`. Then reads the value, increments, rewrites again and then closes the file.

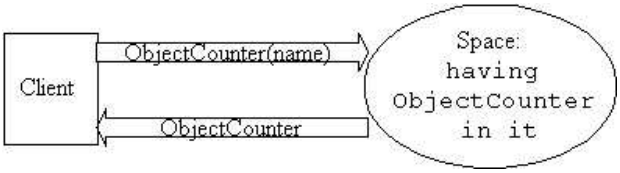


FIGURE 3. The servlet architecture

The response is a Content-type: `application/octet-stream`. It contains the serialization form for an `Integer` object, having the server time in it.

## 5. JavaSpaces: features and specifics

An important class of distributed applications is based on the JINI technology. JINI is the Sun's solution for creating common, everyday, networking appliances that just "plug and work". Fundamentally, JINI is using in an extensive manner the RMI (Remote Method Invocation) [1,12] Java technology. A programming model/infrastructure JINI enables building/deploying of distributed systems organized as *federations of services*. A service can be a hardware or software component, a common channel, a user, a disk drive which can offer a storage service and so on. Once part of a federation, a service can be used by other services or clients

An important particular application of JINI is JavaSpaces [5].

A JavaSpaces server is called *space*, and holds *entries* (typed groups of objects) in it. A distributed Java Spaces application has a space, and a lot of clients that access this space. There are three main operations over space:

**write:** put an object in space using a special Entry object - a container for the objects from space;

**read:** get a copy for an object from space. Look it up using *template* entries, using fields with values (exact matching) and null for wildcards. The object is found by associative methods. If such an object is not in the space, the client waits until such object put in the appearance in the spaces.

**take:** similar with read, find an object from space and move the object from space into the client program.

Therefore, an elegant paradigm for distributed programming and concurrent programming is provided. All operations are transaction security. Entries written to a space are leased (using JINI leasing). For short, a JavaSpaces acts as a "shared memory" for distributed processes. We can preserve any kinds of objects in space, with elaborate "data structures" in them.

For our experiment, the java source of the objects from the spaces is defined as follows:

```
public class ObjectCounter implements Entry {
   public String name;
   public Integer counter;
```

```
    public ObjectCounter() {
    }//ObjectCounter.ObjectCounter
    public ObjectCounter(String name) {
       this.name = name;
    }//ObjectCounter.ObjectCounter
    public ObjectCounter(String name, int counter) {
       this.name = name;
       this.counter = new Integer(counter);
    }//ObjectCounter.ObjectCounter
    public Integer increment() {
       this.counter = new Integer(counter.intValue() + 1);
       return counter;
    }//ObjectCounter.increment
 }//ObjectCounter
```

The `ObjectCounter()` is mandatory for JavaSpaces Technology. The method `ObjectCounter(String name, int counter)` is used for the init part and the method `ObjectCounter(String name)` is repeatedly used for *take* and *rewrite* objects in the space. The JavaSpaces architecture for this application is presented in Figure 4.

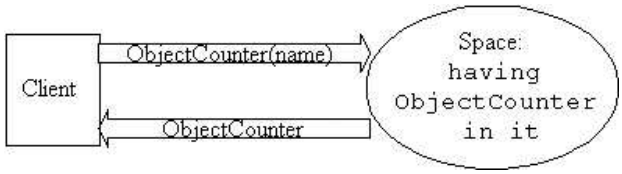


FIGURE 4. The JavaSpaces architecture

The time for a client access includes the operations `setSecurityManager`, construct a `LookupLocator` object, define a `ServiceRegistrar` object and `lookup` the space object. (For details, see [1,6]).

After that, we consider part of the server time, and its current standalone actions are:

```
template = new ObiectContor(NameOfTheCounter);
counter = (ObjectCounter)space.take(template, null, Long.MAX_VALUE);
Integer i = counter.increment();
space.write(counter, null, Lease.FOREVER);
```

## 6. EJB: FEATURES AND SPECIFICS

Enterprise JavaBeans (EJB) [2,8,12] are a container-based component architecture that allow you to easily create secure, scalable and transactional enterprise applications. Developed as session beans, entity beans, or message-driven beans, EJBs are the critical business objects in any J2EE application. The 2.0 version of the specification for EJB introduces important improvements to the bean-managed (BMP) and container-managed (CPM) models for entity persistence.

Our experiment uses the 2.0 CPM model for the counter implementation. The important parts of the Java source excerpt are given below. Firstly, the remote interface is:

```
import javax.ejb.*;
import java.rmi.*;
public interface Counter extends EJBObject {
   int increment() throws RemoteException, FinderException;
}//Counter
```

The home interface is:

```
import java.io.*;
import java.rmi.*;
import javax.ejb.*;
public interface CounterHome extends EJBHome {
   Counter create(String name, int value)
      throws RemoteException, CreateException;
   Counter findByPrimaryKey(String name)
      throws RemoteException, FinderException ;
}//CounterHome
```

The main part of the Entity bean is:

```
import javax.ejb.*;
import java.rmi.*;
public abstract class CounterBean implements EntityBean {
   private EntityContext context;

   public abstract String getName();
   public abstract void setName(String name);
   public abstract int getValue();
   public abstract void setValue(int value);

   public int increment() throws RemoteException, FinderException{
      long ti = System.currentTimeMillis();
      int i = getValue();
      setValue(i+1);
      return((int)(System.currentTimeMillis()-ti));
   }//CounterBean.increment

 - - - - - - - - - - - - - -

}//CounterBean
```

The main part of the standalone client is:

```
 - - - - - - - - - - - - - -
Context initial = new InitialContext(env);
Object ref = initial.lookup("aliasCounter");
CounterHome counterHome = (CounterHome) PortableRemoteObject.narrow(
                                          ref, CounterHome.class);
Counter counter = counterHome.findByPrimaryKey("nameOfCounter");
ts = (long)counter.increment();
 - - - - - - - - - - - - - -
```

The EJB architecture for this application is presented in Figure 5.

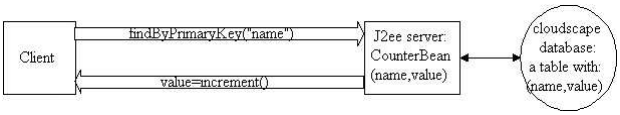


FIGURE 5. The EJB architecture

## 7. TIME COMPARISONS

For comparing these technologies, from the run-time point of view, we made many experiments. At `http://www.cs.ubbcluj.ro/~florin` there is an archive with files having results and program sources. CGI, Servlet and JavaSpaces, codes run in the same conditions, for about 8 hours. EJB was experiment only for 1 hour.

In the following table, we present the run-times after run about 1 minute, 10 minutes, 30 minutes, 1 hour, 4 hours and 8 hours. After each 1000 tests, the client programs verify if the moments of saving results appeared or not. For this reasons, the times for total connections have some differences between our intention to save the results and the real moments to save.

| Min. | Parameter | CGI | Servlet | JavaSpaces | EJB |
|---|---|---|---|---|---|
| 1 | TotalClient | 132901 | 130338 | 151057 | 256569(43171) |
| | TotalServer | 9671 | 93260 | 72045 | 20(22) |
| | MinimumClient | 10 | 0 | 30 | 161(80) |
| | MinimumServer | 0 | 0 | 20 | 0(0) |
| | MaximumClient | 410 | 231 | 540 | 1092(571) |
| | MaximumServer | 398 | 153 | 160 | 10(4) |
| | AverageClient | 14 | 14 | 50 | 256(43) |
| | AverageServer | 1 | 10 | 24 | 0(0) |
| | NumberOfTests | 9000 | 9000 | 3000 | 1000(400) |
| | TotalConnectClient | 132941 | 130398 | 151067 | 256579(43612) |
| 10 | TotalClient | 663214 | 670645 | 694078 | 705275 |
| | TotalServer | 65752 | 471949 | 311208 | 60 |
| | MinimumClient | 10 | 0 | 30 | 150 |
| | MinimumServer | 0 | 0 | 10 | 0 |
| | MaximumClient | 611 | 250 | 2974 | 1092 |
| | MaximumServer | 597 | 173 | 230 | 10 |
| | AverageClient | 15 | 13 | 53 | 235 |
| | AverageServer | 1 | 9 | 23 | 0 |
| | NumberOfTests | 44000 | 49000 | 13000 | 3000 |
| | TotalConnectClient | 663354 | 670805 | 694178 | 705305 |

| | | | | | |
|---|---|---|---|---|---|
| 30 | TotalClient | 1868597 | 1863150 | 1892922 | 2068375 |
| | TotalServer | 266662 | 1310745 | 855115 | 201 |
| | MinimumClient | 10 | 0 | 30 | 150 |
| | MinimumServer | 0 | 0 | 10 | 0 |
| | MaximumClient | 881 | 251 | 2974 | 1742 |
| | MaximumServer | 857 | 246 | 390 | 11 |
| | AverageClient | 15 | 13 | 54 | 229 |
| | AverageServer | 2 | 9 | 24 | 0 |
| | NumberOfTests | 117000 | 136000 | 35000 | 9000 |
| | TotalConnectClient | 1868817 | 1863500 | 1893102 | 2068465 |
| 60 | TotalClient | 3672722 | 3670319 | 3682746 | 3745356 |
| | TotalServer | 433542 | 2611931 | 1656224 | 351 |
| | MinimumClient | 10 | 0 | 30 | 150 |
| | MinimumServer | 0 | 0 | 10 | 0 |
| | MaximumClient | 982 | 321 | 3025 | 2674 |
| | MaximumServer | 961 | 311 | 390 | 11 |
| | AverageClient | 15 | 14 | 54 | 234 |
| | AverageServer | 1 | 10 | 24 | 0 |
| | NumberOfTests | 234000 | 260000 | 67000 | 16000 |
| | TotalConnectClient | 3673022 | 3670959 | 3682996 | 3745536 |
| 240 | TotalClient | 14460504 | 14458049 | 14475715 | |
| | TotalServer | 2015054 | 9953004 | 6532319 | |
| | MinimumClient | 10 | 0 | 30 | |
| | MinimumServer | 0 | 0 | 10 | |
| | MaximumClient | 1001 | 401 | 3085 | |
| | MaximumServer | 984 | 311 | 421 | |
| | AverageClient | 16 | 13 | 54 | |
| | AverageServer | 2 | 8 | 24 | |
| | NumberOfTests | 903000 | 1111000 | 264000 | |
| | TotalConnectClient | 14461685 | 14460173 | 14476456 | |
| 480 | TotalClient | 28558686 | 22400298 | 28861269 | |
| | TotalServer | 3223970 | 14725660 | 12946711 | |
| | MinimumClient | 10 | 0 | 30 | |
| | MinimumServer | 0 | 0 | 10 | |
| | MaximumClient | 1001 | 6399 | 3085 | |
| | MaximumServer | 984 | 6390 | 3034 | |
| | AverageClient | 15 | 13 | 54 | |
| | AverageServer | 1 | 9 | 24 | |
| | NumberOfTests | 1839000 | 1625000 | 527000 | |
| | TotalConnectClient | 28865617 | 29642314 | 28862612 | |

However, EJB performance is very dependent on the underlying configuration. For example, informal tests (`http://www.JBoss.org`) show that on the same PC box, it can run twice as fast under Windows 2000 / Sun JVM than under Linux 2.2 / Sun JVM. Linux users probably already know that linux does not support real

threads. Under heavy load, JBoss will for example crash with 200 concurrent users under linux, whereas it can handle 1000 of them on the same box with Windows 2000. Of course, if you use Apache or Jetty in front of JBoss to handle the thread pooling, this will not be a problem.

In our experiment, EJB randomly crashes on Linux, between 100 to 400 tests. In the first cell from EJB in the above table, we write in brackets, our results after 400 tests.

The main parameter in which we are interested is the *average time client*. **From this point of view, our conclusion is**:

- For this kind of applications, the Servlet technology is optimal. We consider that 13 millisecond is a good average time client.
- The CGI technology have a nearly performance, 15 millisecond is a value nearly of the Servlet technology. The difference is due to the fact that CGI uses much more system resources than Servlet [10].
- JavaSpaces is an elegant solution, but its performance is about 3 times sluggish than Servlet and CGI technologies.
- EJB technology is a robust one, but only if it use a professional EJB server (as JBoss or BEA for example) with a professional operating system (as Solaris, for example).

Thus, for simple distributed applications, with very frequent rate of use but with simple security restrictions we recommend to the use a Servlet or CGI technologies. JavaSpaces and EJB are recommended to be use only for large applications, with a medium frequency of use and with high security and transactional restrictions.

## References

[1] Ayers D. et. al. *Professional Java Server Programming* Wrox Press, 1999
[2] Bodoff S. et.al. *The J2EE Tutorial*. Sun Microsistems, 2001
[3] Boian. F.M. *Distributed programming in Internet* (Romanian) Blue ed. Cluj, Romania, 1998
[4] Breedlove et. al. *Web Programming Unleashed*. Sams.het, 1996, http://sams.mcp.com
[5] Edwards W.K *Core Jini*. Prentice Hall, 1999
[6] Halter S.L. *JavaSpaces Example by Example*. Prentice Hall, 2002, http://www.phptr.com
[7] Kim E. et. al. *CGI Programming Unleashed*. Sams.het, 1996, http://sams.mcp.com
[8] Roman E. *Mastering Enterprinse JavaBeans and the Java 2 platform*Willey, 1999
[9] Umar A. *Object Oriented Client / Server Internet Environments*. Prentice Hall, 1997
[10] * * * http://http://www.coreservlets.com
[11] * * * http://jakarta.apache.org
[12] * * * http://java.sun.com/

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

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLVI**, Number 2, 2001

# FLOW IN NETWORK MODELING TIME TABLING AND SCHEDULING PROBLEM

TEODOR TOADERE

Abstract. In this paper a new model for solving the Time Tabling and Scheduling problem by determining the maximal flow in a specified transportation network is proposed. This transportation network is also related to an assignment problem where $n$-uples must be determined instead of pairs, as with the classical assignment problem.

The network is based on a multipartite graph, whose set of vertices is formed by the source, the sink and subsets with elements to be placed in schedule. The subsets contain: the disciplines, the professors, the class of learners (students, pupils, ...), the classrooms and time duration of the activities.

## 1. Ford-Fulkerson algorithm

We suppose that the Ford-Fulkerson algorithm is well known but we will describe it in Pseudocode. For this purpose we will define the "ecarts" graph attached to a transportation network from the source to the sink, and to a compatible flow.

As it is known, the flow $\phi$ in the network $G = (X, U)$, is called compatible flow if $0 \leq \phi_u \leq c_u$, $\forall u \in U$.

**Definition 1.1.** Let $G = (X, U)$ be a graph. To each compatible flow $\phi = (\phi_1, \ldots, \phi_m)^t$ the "ecarts" graph $G(\phi) = (X, U(\phi))$ is attached. The set of edges (arrows) is built in the following mode: $\forall u = (i, j) \in G$, $u^+ = (i, j) \in U(\phi)$ if $\phi_u < c_u$ and $u^- = (j, i) \in U(\phi)$ if $\phi_u > 0$. The capacities for edges are $c_u - \phi_u$ for $u^+$ and $\phi_u$ for $u^-$.

The capacities for edges from $G(\phi)$ are called residuals capacities. The capacity (i.e. $\min \sum_{e \in W} c(e)$) for each path $w$ from $G(\phi)$ is called the path residual capacity.

The following result based on Ford-Fulkerson algorithm is well known. The result may help us to find out a maximal flow in a transportation network.

**Theorem 1.1.** [1] Let $\bar{\phi}$ be a flow from the source $s$ to sink $t$, in a transportation network. Let $G(\phi)$ be, also, the attached "ecarts" graph. Then, the flow $\bar{\phi}$ is maximal if and only if there is no path from source $s$ to sink $t$, in $G(\phi)$.

**Algorithm 1.1.** Ford-Fulkerson algorithm. [2]:

---

2000 *Mathematics Subject Classification.* 05C38.

1998 *CR Categories and Descriptors.* G.2.2 [**Mathematics of Computing**] : Discrete Mathematics – *Graph Theory*; Path and Circuit Problems.

(a) Initializations;
$k$:=0;
Suppose $\phi^0$ a compatible flow, e.g. $\phi^0 = (0, 0, \ldots, 0)$.
(b) Halt criterium:
If $\exists \mu^k$ a path from $s$ to $t$ in $G(\phi^k)$
Then goto $(c)$
Else stop; $\phi^k$ is optimal flow.
(c) Let $\varepsilon$ be the residual capacity for $\mu^k$;

$$\phi_u^{k+1} = \begin{cases} \phi_u + \varepsilon, & u^+ \in \mu^k \text{ or } u = 0 \\ \phi_u - \varepsilon, & u^- \in \mu^k, \text{ for } u \in U \\ \phi_u, & \text{otherwise} \end{cases}$$

$k$:=$k+1$;
goto $(b)$.

For the existence of $\mu^k$, a path in the "ecarts" graph $G(\phi^k)$ and for its determination (and also for determination of the value of $\varepsilon$) we may use two procedures [2]:

$p_1$) From $G(\phi^k)$ a tree is built such that the root is $s$ and the other vertices are the vertices from $G(\phi^k)$ . We may use a procedure to mark the vertices and then we may find out the residual capacity for a path form $s$.
$p_2$) Without building the "ecarts" graph $G(\phi^k)$:
Each $j \in X$ has a label $(e_1, e_2, e_3)$ such that: $e_1 \in X \cup \{0\}$; $e_2 \in \{+, -\}$; $e_3 \in \mathbb{R}_+$, with the following meaning:

- if $e_2 = +$ and $e_1 = i$, then $\exists u = (i, j) \in G$ and $\exists \mu \in G(\phi^k)$, path from $s$ to $j$ and the last edge is $u^+ = (i, j)$, its capacity is $e_3$.
- if $e_2 = -$ and $e_1 = i$, then $\exists u = (j, i) \in G$ and $\exists \mu \in G(\phi^k)$, path from $s$ to $j$ and the last edge is $u^- = (i, j)$, its capacity is $e_3$.

The source $s$ is labeled with $(0, +, maxint)$. The main iteration of this "labeled" procedure consists in:

repeat
$k$:=0;
for each labeled $i \in X$ do
if $\exists j \in X$ (not labeled) such that $u = (i, j) \in U$ and $\phi_u < c_u$,
then one assigns to $j$ the label $(i, +, min\{e_3(i), c_u - \phi_u\})$; $k$:=1;
endif;
if $\exists j \in X$ (not labeled) such that $u = (j, i) \in U$ and $\phi_u > 0$,
then one assigns to $j$ the label $(i, -, min\{e_3(i), \phi_u\})$; $k$:=1;
endif;
endfor;
until ($t$ is labeled) or ($k$=0);

If $t$ is labeled at the end of this algorithm then we find the path $\mu^k = (d_p, d_{p-1}, \ldots, d_1)$, from $s = d_p$ to $t = d_1$, using the values of $e_1$. One starts from $e_1(t)$, so:

$p := 1$;
$d_1$:= t;
while $d_p \neq s$ do
    $p$:= $p + 1$;
    $d_p$:=$e_1(d_{p-1})$;
endwhile;

The final value of $\varepsilon$ is equal to $e_3(t)$.
The other final condition, $k = 0$, shows us that there is no path in $G(\phi^k)$.

## 2. Flow in Network Modeling Time Tabling and Scheduling Problem

In few words the Time Tabling and Scheduling Problem is:

**Data:**
- the set of disciplines;
- the set of professors;
- the set of classes of learners (students, pupils, ...);
- the set of classrooms (rooms);
- the days and the time periods for activities planning;
- the dependences between the all five previous sets.

**Requirements:** A schedule than every element from every set must be well found:
- all of disciplens must be planned;
- every professor can realize the weekly hours number;
- all of classes (students, pupils, ...) must be in schedule;
- all of rooms must be optimally used.

A schedule can be interpreted as a string of information (or as a database). From it may be extracted:
- the schedule of all classes (students, pupils, ...);
- the schedule of all professors;
- the schedule of all rooms.

A record (a position in this schedule) must contain: the date, the time, the discipline, the room, the class and the professor.

Suppose that we can encode the time periods of a week. For example: suppose that all activities (courses, seminars, laboratories, ...) last two hours (120 minutes). One can count the pair hours between 8 and 20 from Monday to Friday. One count 5 days x 6 period = 30 values (the attribute **H**our). On the other hand the class of students, pupils, ... can be encoded (the attribute **C**lass). A similar codification can be done for each discipline (the attribute **D**iscipline). From each pair Class-Discipline one identifies the professor.

This activity of making the schedule is modeling like a maximal flow problem in a special graph. Than, the elements of the previous sets will be considered as vertices. Will be five types of vertices:
- Disciplines;
- Professors;

- Class (students, pupils, ...);
- Rooms;
- Hours.

A record (a position in the schedule) can be interpreted as a path in this special graph and contains vertices from every type.

The first proposed model attaches to the Time Tabling and Scheduling Problem a multipartite graph. This graph has five partitions:

- the first partition contains the disciplines;
- the second partition contains the professors;
- the third partition contains the classes (students, pupils,);
- the fourth partition contains the classrooms;
- the final partition contains the hours.

In this case for each professor, each discipline, each class, each room there is a corresponding vertex in that graph. For each hour (or two hours = 120 minutes) we have a vertex in graph in the hours partition as, for example:

- Monday: the time 10-12 is labeled with 2;
- Wednesday: the time 14-16 is labeled with 16, and so on.

There will be 30 vertices (5 days x 6 periods of time / each day). If we would like to solve the Time Tabling and Scheduling Problem for a school of pre-university level one can consider 60 vertices for 60 hours (5 days x 12 periods of time / each day).

The vertices with compatibilities with some activities will be connected. For example, every discipline is taught by a subset of the set of professors, every professor do some activities only with certain classes of students, every room must be available at certain hours weekly.

In addition to the five partitions of vertices, we have two vertices: $s$ for source and $t$ for sink. An example is shown in Figure 1.

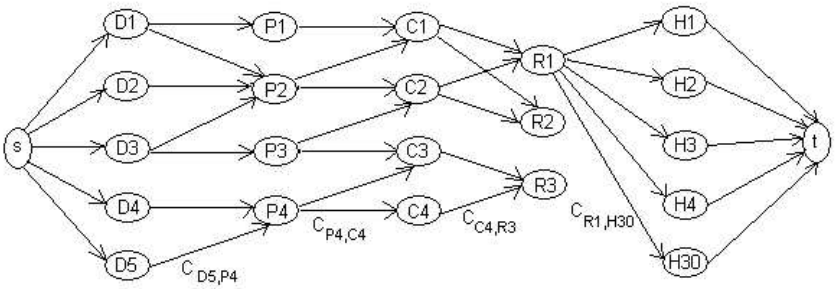


FIGURE 1. A multipartite graph example

As we see, every discipline is connected to a subset of professors-vertices. For example: discipline $D_1$ can be taught only by professors $P_1$ and $P_2$; the capacity value between vertices $D_5$ and $P_4$ is $c_{D_5P_4}$.

Every professor can teach a subset of classes of learners. For example: Professor $P_4$ may teach only classes $C_3$ and $C_4$; the capacity value between vertices $P_4$ and $C_4$ is $c_{P_4C_4}$.

Every class of learners may carry its activities in certain rooms. For example: $C_1$ carries its activities only in $R_1$ and $R_2$. The capacity value between vertices $C_4$ and $R_3$ is $c_{C_4R_3}$.

Finally, every classroom will be connected to the available hours. For example: $R_1$ is related to $H_1, \ldots, H_{30}$; the capacity value between vertices $R_1$ and $H_{30}$ is $c_{R_1H_{30}}$.

The source $s$ is connected to every discipline-vertex and the sink $t$ is connected to every hour-vertex.

Denotation:

| | | |
|---|---|---|
| $G = (V, E)$ | – | the network (the graph attached to the problem); |
| $V$ | – | the set of vertices; |
| $E$ | – | the set of edges; |
| $R = (G, s, t, c)$ | – | the transportation network; |
| $s$ | – | the source; |
| $t$ | – | the sink; |
| $c$ | – | $c : E \to \mathbb{R}_+$, $c(i)$ = the capacity of edge $i$, $\forall i \in E$. |

The capacities will be built such that:

- if a professor $P$ must teach $c$ hours for a discipline $D$ then the edge between $D$ and $P$ has the capacity equal to $c$ ($c_{D,P} = c$);
- if a professor $P$ must do $c$ activity hours with a class $C$ then the edge between $P$ and $C$ has the capacity equal to $c$ ($c_{P,C} = c$);
- for every edges between a class $C$ and a classrooms $R$, the capacity is at least $\sum_{e \in \Gamma^{-1}(C)} c(e)$; the maximal may be either 30 or 60;
- the capacities attached to the edges between classrooms-vertices and hours-vertices will be set to 1 or 2;
- every edge form source $s$ to discipline $D$ has the capacity $c_{s,D}$ equal to the number of weekly-hours;
- the edges between every hour $H$ and the sink vertex $t$ have the capacity $c_{H,t} = 1$.

An example with one discipline ($D_1$), two professors ($P_1$, $P_2$), two classes ($C_1$, $C_2$), two classrooms ($R_1$, $R_2$) and three hours ($H_1$, $H_2$, $H_3$) is depicted in Figure 2.

Every path from $s$ to $t$ may be a position in schedule because the path locates the discipline, the professor, the class, the classroom and the time.

Now, for the schedule determination we solve the problem with Ford-Fulkerson algorithm, by determining the maximal flow in the network $R = (G, s, t, c)$. The maximal flow between $s$ and $t$ is found out by successive selection of certain paths in

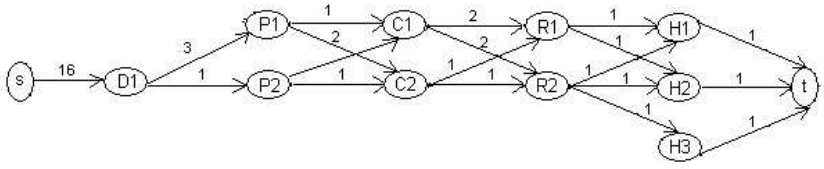


FIGURE 2. A specific example

the ecarts graph. Every selected path from $s$ to $t$ is a position in the schedule. For every edge of these paths, the capacity equals to the capacity minus the residual capacity.

When obtaining the maximal flow in the schedule, all the schedule positions have been set. Now the schedule is realized.

## 3. Conclusions and Improvements

By using the method presented above a schedule variation is determined, which depends on the selection order of the path in the ecarts graph.

If an optimization of the produced schedule is needed, we suggest that to the transportation network defined above some extra edges, called inhibitory edges, should be added. These edges define the schedule restrictions, e.g. if a certain professor is not available at specific hours, an inhibitory edge will be added between those professor-vertex and hour-vertex. As another example, if some classes, some activities may not take place, an inhibitory edge will be added between the corresponding vertices. When using the Ford-Fulkerson algorithm, the paths with inhibitory edges will not be selected. Then, the network will be represented by using the two types of edges, e.g. two adjacent matrices or a matrix with values equal the edges capacity, or zero, if no edge exists, or -1 for an inhibitory edge. Each solver may decide on the data representation of the network defined by the model described for the scheduling problem.

## References

[1] Ford, L.R. Jr., *Network-flow theory*, The Rand Corporation, 1956, pp. 293
[2] Toadere, T., *Grafe: Teorie, algoritmi şi aplicaţii*, Editura Albastră, Cluj, 2001

Department of Computer Science, "Babeş-Bolyai" University, 1 M. Kogălniceanu St., RO-3400 Cluj-Napoca, Romania
*E-mail address*: toadere@cs.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLVI**, Number 2, 2001

# PRINCIPAL COMPONENTS ANALYSIS BASED ON A FUZZY SETS APPROACH

HORIA F. POP

Abstract. As with any other multivariate statistical method, Principal Components Analysis is sensitive to outliers, missing data, and poor linear correlation between variables due to poorly distributed variables. As a result data transformations have a large impact upon PCA. This paper introduces a powerful approach to improve PCA: robust fuzzy PCA algorithm (FPCA). The matrix data is fuzzified, thus diminishing the influence of the outliers.

## 1. Introduction

Several statistical methods for the analysis of large quantities of data have been applied to scientific problems during the last decades. One of these methods, principal component analysis (PCA) showed special promise for furnishing new and unique insights into the data interactions.

PCA is designed to reduce the number of variables that need to be considered to a small number of indices (axes) called the principal components, that are linear combinations of the original variables. The new axes lie along the directions of maximum variance such that containing most of the information. PCA provides an objective way of finding indices of this type so that the variation in the data can be accounted for as concisely as possible.

In the case of an $n$-dimensional problem, often the number of components needed to describe, say 90% of the sample variance is less than $n$, so that PCA essentially affords one a technique whereby the dimensionality of the variable space can be reduced, i.e., it is a dimension reduction method. It may well turn out that usually two or three principal components provide a good summary of all the original variables. Moreover, PCA offers a second important tool for multidimensional analysis that derives, in fact, from its original application in the social sciences and from which it took its name. In other words, PCA can also reveal those underlying factors or combinations of the original variables that principally

2000 *Mathematics Subject Classification.* 68N30.

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

determine the structure of the data distribution and that not infrequently are related to some real influencing factors in the sample population. An important issue in PCA is the interpretation of components, to help determine after the reduction of the observation space, which initial variables has the greatest shares in the variance of particular principal components. This information can be obtained using coefficients of determination (loadings) established between the components and the initial variables.

## 2. Principal Components Analysis (PCA)

PCA is based on eigenanalysis of the covariance or correlation matrix. Let us consider a data set $X = \{x^1, \ldots, x^p\}$, and its covariance matrix $M$:

$$M_{ij} = \frac{1}{p-1} \sum_{k=1}^{p} (x_i^k - \bar{x}_i) \cdot (x_j^k - \bar{x}_j), \ i, j = 1, \ldots, n. \tag{1}$$

Let us also consider the ortonormal eigenvectors $e^i$ of the matrix $M$, and the corresponding eigenvalues $\lambda_i$ $(i = 1, \ldots, n)$.

The principal components of the data set $X$ appear as linear combinations of the original variables in the form

$$\mathbf{PC_i} = e_1^i y^1 + e_2^i y^2 + \cdots + e_n^i y^n, \tag{2}$$

where $y^i$ represents the $i$-th original variable $(y_j^i = x_i^j)$, and $e_j^i$ represent the $j$-th element of the eigenvector $e^i$ of the matrix $M$.

A constraint that $(e_1^i)^2 + (e_2^i)^2 + \cdots + (e_n^i)^2 = 1$ is imposed on all components. The constraint is introduced in order to ensure that $Var(\mathbf{PC}_i)$ cannot be increased by simply increasing any of the $e_j^i$ values.

From the orthonormality of $e^1, e^2, \ldots, e^n$ it follows that

$$\begin{array}{ll} e^{iT} \cdot e^i = 1, & \text{for any } i \in \{1, \ldots, n\} \\ e^{iT} \cdot e^j = 0, & \text{for any } i, j \in \{1, \ldots, n\}, i \neq j \\ e^{iT} \cdot M \cdot e^i = 1, & \text{for any } i \in \{1, \ldots, n\} \\ e^{iT} \cdot M \cdot e^j = 0, & \text{for any } i, j \in \{1, \ldots, n\}, i \neq j, \end{array} \tag{3}$$

and

$$M = \lambda_1 e^1 e^{1T} + \lambda_2 e^2 e^{2T} + \cdots + \lambda_n e^n e^{nT}. \tag{4}$$

where $^T$ denotes the transposing operation.

The basic property of the new variables is their lack of correlation. We have that

$$\mathrm{Var}(e^i X) = \lambda_i, \ \text{for } i = 1, \ldots, n \tag{5}$$

and

$$\mathrm{Cov}(e^i X, e^j X) = 0, \ \text{for } i, j = 1, \ldots, n, i \neq j. \tag{6}$$

The first principal component $\mathbf{PC}_1$ is that linear combination of sample values for which the "scores" have maximum variation. The second component $\mathbf{PC}_2$ has scores that are uncorrelated with the scores for $\mathbf{PC}_1$. Among the many linear combinations with this property we select the one which has maximum variation among its scores. The third component $\mathbf{PC}_3$ is defined to be that linear combination which has the maximum variation among all those combinations whose scores are uncorrelated with the scores of the first two components. Subsequent components are defined analogously.

Principal component analysis as any other multivariate statistical methods are sensitive to outliers, missing data, and poor linear correlation between variables, due to poorly distributed variables. As a result, data transformations have a large impact upon PCA [3].

One of the most illuminating approach to robustify PCA appears to be the fuzzification of the matrix data by diminuishing in this way the influence of the outliers.

## 3. Fuzzy Principal Components Analysis (Fuzzy PCA)

Fuzzy clustering is an important tool to identify the structure in data. In general, a fuzzy clustering algorithm with objective function can be formulated as follows: let $X = \{x^1, \ldots, x^n\} \subset \mathbb{R}^p$ be a finite set of feature vectors, where $n$ is the number of objects (measurements) and $p$ is the number of original variables, $x_k^j = [x_1^j, x_2^j, \ldots, x_p^j]^T$ and $L = (L^1, L^2, \ldots, L^s)$ be a $s$-tuple of prototypes (supports) each of which characterizes one of the $s$ clusters; a partition of $X$ into $s$ fuzzy clusters will be performed by minimizing the objective function [2]:

$$J(P, L) = \sum_{i=1}^{s} \sum_{j=1}^{n} (A_i(x^j))^m d^2(x^j, L^i),$$

where $P = \{A_1, \ldots, A_s\}$ is the fuzzy partition, $A_i(x^j) \in [0, 1]$ represents the membership degree of feature point $x^j$ to fuzzy cluster $A_i$, $m > 1$ is the fuzziness index, and $d(x^j, L^i)$ is the distance from the feature point $x^j$ to the prototype of the cluster $A_i$. If $L_i$ are defined as points in the $\mathbb{R}^p$ Euclidean space, the distance $d$ may be defined as the Euclidean distance.

According to the choice of prototypes and the definition of the distance measure, different fuzzy clustering algorithms are obtained. If the prototype of a cluster is a point — the cluster center — it will produce spherical clusters; if the prototype is a line it will produce tubular clusters, and so on. Also, elements with a high degree of membership in the $i$-th cluster (i.e. close to the cluster's center) will contribute significantly to this weighted average, while elements with a low degree of membership (far from the center) will contribute almost nothing.

Due to the problem at hand, we will consider that the fuzzy set is characterized by a linear prototype, denoted $L(u, v)$, where $v$ is the center of the class and $u$, with

$\|u\| = 1$, is the main direction. This line is also called *the first principal component* of the set, and its direction is given by the unit eigenvector $u$ associated with the largest eigenvalue $\lambda_{max}$ of, for example, the covariance matrix $C = (C_{ij})$, formed by the elements

$$C_{ij} = \frac{\sum_{k=1}^{p} A(x^k)^m \cdot (x_i^k - \bar{x}_i) \cdot (x_j^k - \bar{x}_j)}{\sum_{k=1}^{p} A(x^k)^m}, \ i, j = 1, \ldots, n. \tag{7}$$

where $\bar{x}_i$ is the arithmetic mean of the $i$-th variable, $m > 1$ is the fuzziness index. The settings above mean that the fuzzy set $A$ is characterized by the linear prototype $\mathbf{PC}_1$ produced considering the fuzzy covariance matrix $C$.

We wish to determine the particular membership degrees $A(x)$ such that the first principal component is best fitted along the items of the data set $X$. The algorithm proposed in this paper is a natural extension of the Fuzzy 1-Lines Algorithm [5].

Let us denote by $\alpha$ the membership degree corresponding to the farthest outlier. For the moment we consider that $\alpha$ is a value preset by the user. The membership degrees $A(x)$ will be produced using the following mechanism:

**Algorithm** DETERMINE_FUZZY_MEMBERSHIPS($\alpha$):

(1) Initialize $A(x) = 1$, for all $x \in X$;

(2) Determine the linear prototype $L(u, v)$: $u$ is the eigenvector corresponding to the largest eigenvalue of the matrix $C$ computed as in (7); $v$ is the weighting center of the fuzzy cluster $A$, weighted by the $m$-th power of the membership degrees:

$$v = \frac{\sum_{j=1}^{n} A(x^j)^m \cdot x^j}{\sum_{j=1}^{n} A(x^j)^m};$$

(3) Determine the new fuzzy membership degrees $A(x^j)$:

$$A(x^j) = \frac{\frac{\alpha}{1-\alpha}}{\frac{\alpha}{1-\alpha} + \left(d^2(x^j, L)\right)^{\frac{1}{m-1}}};$$

(4) if the new fuzzy set is close enough to the old one, then Stop and return the new fuzzy set; else go to Step 2.

The algorithm suggested above depends on the input variable $\alpha$. As opposed to the general case, we now do have a way to determine the best value for $\alpha$. Of

course, we are interested to find fuzzy membership degrees that contribute to producing a better fitted first principal component along the data set. But, since the eigenvalue associated to a principal component describes the scatter of data along that component, we are also interested in producing a first principal component characterised by an eigenvalue that is as large as possible. As a consequence, we will prefer that particular value of $\alpha$ that maximizes the eigenvalue associated to the first principal component.

Because of the fact that we are interested in real-world applications of this algorithm, an exact value of $\alpha$ is not required. Instead, we will simply work through a loop between 0 and 1, with a step to be chosen by the user, and select the value of $\alpha$ that maximizes our criterion. The produced algorithm follows:

**Algorithm** DETERMINE_BEST_ALPHA():

(1) Initialize *step* as appropriate; initialize $\alpha_0 = 0$ and $\lambda_0 = 0$;
(2) Set $\alpha = step$, the first value to be considered;
(3) Call DETERMINE_FUZZY_MEMBERSHIPS($\alpha$) with the current value of $\alpha$, and determine the optimal fuzzy membership degrees $A(x)$;
(4) Using the fuzzy membership degrees determined above, compute the matrix $C$ as in (7), and compute the eigenvalue $\lambda$ corresponding to its largest eigenvector (i. e. the first principal component);
(5) If $\lambda > \lambda_0$ then set $\lambda_0 = \lambda$ and $\alpha_0 = \alpha$;
(6) Increment $\alpha$ by *step*; if $\alpha < 1$ then resume from Step 3; else stop, and return $\alpha_0$ as the optimal value for $\alpha$.

Now we have all the prerequisites for writing the algorithm. We will call this algorithm **Fuzzy (first component) Principal Component Analysis (FPCA)**:

**Algorithm** FPCA():

(1) Determine the optimal value of $\alpha$ by calling DETERMINE_BEST_ALPHA();
(2) Call DETERMINE_FUZZY_MEMBERSHIP($\alpha$) with the value of $\alpha$ computed above, and determine the optimal value of the fuzzy membership degrees;
(3) Using the fuzzy membership degrees determined above, compute the matrix $C$ as in (7), and compute its eigenvalues and eigenvectors; these are the fuzzy principal components and the corresponding scatter values.

## 4. Experiments

We have selected for our experiments the set of 48 Roman pottery sherds presented in [1] and analysed in [4].

### 4.1. **PCA on Roman pottery data.**

The principal components produced using Classical PCA on Roman pottery data are depicted in Table 1, together with their associated eigenvalues.

Based on these values, we build reduction coefficients corresponding to different dimensionality reduction criteria. These reduction coefficients show the amount

of original information explained by keeping only a limited number of variables or principal components, and are depicted in Table 2.

| Eigenvalue | Eigenvector | | | | | | |
|---|---|---|---|---|---|---|---|
| 3.04969 | -0.288946 | -0.523259 | 0.352801 | 0.32056 | -0.410301 | -0.466635 | 0.171429 |
| 2.13202 | -0.250928 | -0.0121934 | 0.450562 | -0.49078 | 0.306824 | -0.301171 | -0.555131 |
| 0.906438 | 0.787617 | -0.104202 | 0.285086 | -0.320179 | 0.0265746 | -0.278653 | 0.326587 |
| 0.530287 | -0.206816 | -0.336004 | 0.218458 | 0.0857632 | 0.715959 | 0.266077 | 0.453712 |
| 0.193831 | 0.0999946 | 0.550648 | 0.651976 | 0.491662 | 0.00155423 | 0.136672 | -0.0360862 |
| 0.135622 | 0.421228 | -0.496785 | -0.0255271 | 0.402459 | 0.144139 | 0.209852 | -0.590197 |
| 0.0521156 | -0.0547764 | -0.228659 | 0.343225 | -0.377891 | -0.451031 | 0.693099 | 0.0171312 |

TABLE 1. Loadings of the principal components and their associated eigenvalues, for the classical PCA

| Variables | Eigenvalue | Successive difference | Proportion | Cummulative proportion |
|---|---|---|---|---|
| 1 | 3.04969 | 0.917665 | 0.435669 | 0.435669 |
| 2 | 2.13202 | 1.22558 | 0.304574 | 0.740244 |
| 3 | 0.906438 | 0.376152 | 0.129491 | 0.869735 |
| 4 | 0.530287 | 0.336456 | 0.0757552 | 0.94549 |
| 5 | 0.193831 | 0.0582092 | 0.0276901 | 0.97318 |
| 6 | 0.135622 | 0.083506 | 0.0193745 | 0.992555 |
| 7 | 0.0521156 | 0.0521156 | 0.00744509 | 1 |

TABLE 2. Reduction coefficients for the classical PCA

4.2. **Fuzzy PCA on Roman pottery data.** The principal components produced using Fuzzy PCA on Roman pottery data are depicted in Table 3, together with their associated eigenvalues. The optimal value of the $\alpha$ index has been determined to be 0.01.

Based on these values, we build reduction coefficients corresponding to different dimensionality reduction criteria. These reduction coefficients show the amount of original information explained by keeping only a limited number of variables or principal components, and are depicted in Table 4. The scores of the first two principal components are displayed in Figure 1.

By comparing Tables 1 and 3, we remark a larger value for the first eigenvalue as computed in the case of the Fuzzy PCA method. This shows an ability of the Fuzzy PCA method to get a better fit for the first principal direction among the data set.

A similar conclusion may be drawn by an analysis of Tables 2 and 4, with respect to the different reduction coefficients. For example, the cummulative proportion is 0.435669, 0.740244 and 0.869735 (for the first, the first two, and the first three variables, respectively) in the case of classical PCA, and 0.952995, 0.967357 and

| Eigenvalue | Eigenvector | | | | | | |
|---|---|---|---|---|---|---|---|
| 5.10731 | 0.0282576 | 0.524877 | -0.458777 | -0.279638 | 0.299205 | 0.4783 | -0.341669 |
| 0.076967 | 0.858295 | 0.0479117 | 0.181638 | 0.198671 | 0.0109479 | 0.128092 | 0.414782 |
| 0.0561507 | -0.189394 | -0.166621 | 0.0940642 | 0.121235 | 0.885688 | 0.0541047 | 0.354191 |
| 0.0439734 | -0.396615 | 0.5612 | 0.0312381 | 0.267516 | -0.267034 | 0.183698 | 0.591742 |
| 0.0311047 | 0.25096 | 0.592121 | 0.594966 | -0.0673231 | 0.206804 | -0.390974 | -0.17963 |
| 0.0286412 | -0.0314195 | 0.079254 | -0.183425 | 0.877202 | 0.0802429 | -0.0962796 | -0.417007 |
| 0.0150737 | 0.0734132 | -0.150373 | 0.599232 | 0.148506 | -0.0734878 | 0.745667 | -0.171598 |

TABLE 3. Loadings of the principal components and their associated eigenvalues, for FPCA

| Variables | Eigenvalue | Successive difference | Proportion | Cummulative proportion |
|---|---|---|---|---|
| 1 | 5.10731 | 5.03034 | 0.952995 | 0.952995 |
| 2 | 0.076967 | 0.0208163 | 0.0143616 | 0.967357 |
| 3 | 0.0561507 | 0.0121773 | 0.0104774 | 0.977834 |
| 4 | 0.0439734 | 0.0128687 | 0.00820519 | 0.986039 |
| 5 | 0.0311047 | 0.00246344 | 0.00580395 | 0.991843 |
| 6 | 0.0286412 | 0.0135676 | 0.00534429 | 0.997187 |
| 7 | 0.0150737 | 0.0150737 | 0.00281266 | 1 |

TABLE 4. Reduction coefficients for FPCA

0.977834 in the case of fuzzy PCA. This shows a better capability to concentrate the more information in less principal components, for the case of fuzzy PCA.

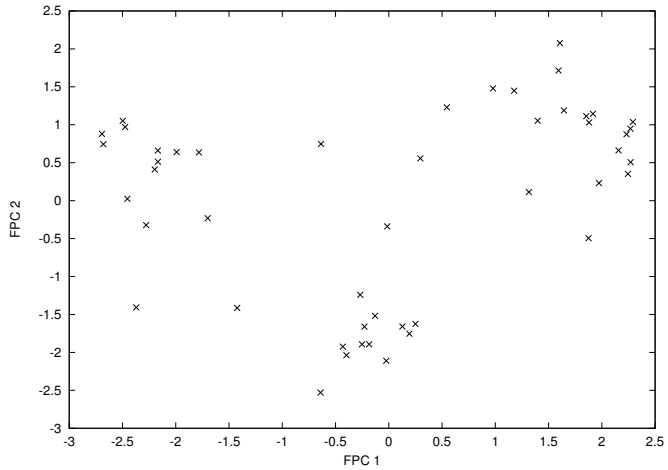


FIGURE 1. Scores of the first two principal components of Roman pottery data using FPCA

## 5. Conclusions

A fuzzy principal component analysis (FPCA) method for robust estimation of principal components has been described in this paper. The efficiency of the new algorithm was illustrated on a data set of 48 Roman pottery sherds. The FPCA method achieved better results mainly because it is more compressive than classical PCA. For the case of a two component model, FPCA accounts for 96.74% of the total variance, and PCA accounts only for 74.02%. Since much more classical principal components would be needed to account for the same total variance as two fuzzy principal components, the fuzzy PCA becomes a much more desirable data analysis tool.

This, together with a sharper data separation, encourages the further research on fuzzy principal components analysis, as well as the fuzzification of other important data analysis techniques.

## References

[1] Aruga, R., Mirti, P., Casoli, A., Application of Multivariate Chemometric Techniques to the Study of Roman Pottery (Terra Sigillata), *Anal. Chim. Acta 276* (1993), 197–205.

[2] Dumitrescu, D., Sârbu, C., Pop, H. F., A Fuzzy Divisive Hierarchical Clustering Algorithm for the Optimal Choice of Sets of Solvent Systems, *Anal. Lett. 24* (1994), 1031–1054.

[3] Hubert, M., Rousseeuw, P. J., Verboven, S. A, Fast Method for Robust Principal Components with Applications to Chemometrics, *Chemom. Intell. Lab. Syst. 60* (2002), 101–111.

[4] Pop, H. F., Dumitrescu, D., Sârbu, C., A Study of Roman Pottery (terra sigillata) Using Hierarchical Fuzzy Clustering, *Anal. Chim. Acta 310* (1995), 269–279.

[5] Pop, H. F., Sârbu, C., A New Fuzzy Regression Algorithm, *Anal. Chem. 68* (1996), 771–778.

Department of Computer Science, Babeş-Bolyai University, 1 M. Kogălniceanu St., RO-3400 Cluj-Napoca, Romania

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

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLVI**, Number 2, 2001

# SEMANTICS FOR CONSTRAINED AND RATIONAL DEFAULT LOGICS

MIHAIELA LUPEA

Abstract. The default nonmonotonic reasoning was formalised by a class of logical systems: default logics (classical, justified, constrained, rational), based on the same syntax which utilises nonmonotonic inference rules: defaults, but with different semantics for the defaults. In this paper we introduce a uniform semantic characterisation for the constrained and rational extensions of a default theory. This characterisation is an operational approach of the nonmonotonic reasoning that is viewed as a successive application of the applicable defaults. During the reasoning process can be observed the interaction between the defaults and the reasoning context. The graphical interpretation associated to the semantic characterisation of extensions illustrates the type of applicability: *cautious* (for constrained extensions) and *hazardous* (for rational extensions) of the defaults and some formal properties: semi-monotonicity, regularity, existence of extensions, commitment to assumptions of these variants of default logic.

## 1. Introduction

An important part of commonsense reasoning is default reasoning, which means drawing conclusions in the absence of complete information using default assumptions. This type of reasoning is nonmonotonic because the conclusions (formulas which are only plausible, not necessarily true) inferred can be later invalidated by adding new facts. Reiter [6] introduced nonmonotonic inference rules called *defaults*, which permit reasoning on the basis of "the lack of evidence to the contrary". The *classical default logic* was the first logical system that formalizes the default nonmonotonic reasoning.

A *default theory* is a pair (D, W), where W is a set of consistent formulas from first order logic and D is a set of default rules. W contains the facts (axioms) of the theory and D contains general rules that might have exceptions. A *default rule* has the form[1]:$d = \frac{\alpha:\beta}{\gamma}$, where $\alpha, \beta, \gamma$are formulas of first order logic,$\alpha$ is the *prerequisite* (*Prereq(d)*) of the default d, $\beta$ is the *justification (Justif(d))* of the default d and $\gamma$ is the *consequent (Conseq(d))* of the default d.

In this paper the followin g notations will be used: Justif(D)=$\bigcup_{d\in D} Justif(d)$, Prereq(D)= $\bigcup_{d\in D}$ Pr$ereq(d)$, Conseq(D)= $\bigcup_{d\in D} Conseq(d)$.

---

1991 *Mathematics Subject Classification.* 03B70, 68T27, 68T37.

1998 *CR Categories and Descriptors.* I2 [**Artificial Intelligence**] Deduction and Theorem Proving − nonmonotonic reasoning and belief revision.

[1]Due to the (semi) representability results for these versions of default logic, we use in this paper only defaults with at most one justification (unitary default theories).

A default $d = \frac{\alpha:\beta}{\gamma}$ can be applied and thus derive $\gamma$ if $\alpha$ *is believed* and *it is consistent to assumed* $\beta$.

Using the classical inference rules and the defaults we can extend the initial set of facts with new formulas called *nonmonotonic theorems* obtaining *extensions*.

**Definition 1.1[6]:** Let (D,W) be a default theory. For any set of formulas S, let $\Gamma$(S) be the smallest set of formulas S' such that:

(1) W$\subseteq$S';
(2) Th(S')= S',whereTh(X) is the set of all the theorems obtained from the set X of formulas and using the classical inference rules;
(3) For any $\frac{\alpha:\beta}{\gamma} \in D$, if $\alpha \in$S' and $\neg\beta \notin$S then $\gamma \in$S'. The application of the default rule means: if we can believe $\alpha$and $\neg\beta$ is not believed, then we can believe $\gamma$.

A set E of formulas is a *classical extension* of (D,W) if and only if $\Gamma$(E)=E.

A classical extension for a default theory is a maximal set of conclusions (beliefs) derived from the facts of W using classical derivation and the defaults as inference rules. A default theory may has zero, one or more classical extensions.

Due to the individual consistency checking of justifications and thus the loss of implicit assumptions when are constructed the classical extensions, this logical system does not satisfy some desirable formal properties: semi-monotonicity, regularity, existence of extensions, commitment to assumptions. In classical default logic these properties are satisfied only for normal default theories, that are the theories with all the defaults of the form: $\frac{\alpha:\beta}{\beta}$.

The next versions of default logic (justified default logic, constrained default logic, rational default logic) try to obtain for general default theories the above properties by modifying the meaning of the statement "*it is consistent to assumed* $\beta$**"**.

Reiter has not provided a semantic for classical default logic, but he has observed that the application of defaults restricts the models of the initial set of facts W, and thus the class of models of an extension is a restricted class of models of W. This idea was formalised by Lukaszewicz [2] for normal default theories and then generalised by Etherington in [1]. Later on, as the new versions of default logics were defined, new approaches of the semantic of these logics appeared.

The *focused models semantics* was introduced for the classical default logic in [1] and then used in [7] for constrained default logic. This semantics is based on a preference between *focused models structures* induced by some set of default rules. An extension is characterized by a maximal models structure.

In the papers [7, 8] Schaub has developed a uniform semantical framework for all the variants of default logic. This approach is called *possible worlds semantics for default logics* and uses Kripke structures to characterise the two components of an extension: the set of beliefs and the underlying assumptions.

The aim of this paper is to provide a semantic characterization for constrained extensions and rational extensions of a default theory in the spirit of the approach of Lukaszewicz [3]. Thus we can obtain an uniform approach of the semantics for classical, justified, constrained, rational default logics, based on the idea that the reasoning process, viewed as a successive application of the applicable defaults,

restricts the models of the initial set of facts. The advantage of this characterization is its graphical interpretation, which illustrates the semantics of applicability conditions for defaults and some formal properties.

The paper is organized as follows. Section 2 provides some notions and results about constrained and rational default logics. In Section 3 we propose an operational semantic characterization for constrained and rational extensions. Section 4 of the paper contains the graphical interpretation of the semantic characterization for constrained and rational extension. The last section is a comparative study of the formal properties of the variants (classical, justified, constrained, rational) of default logic from a semantical point of view.

## 2. Constrained and rational defaults logics

*Schaub defined constrained default logic* in [7] as an alternative approach to classical default logic. The nonmonotonic reasoning formalized by this logic is based on the observation that when we draw conclusions, we have to keep track of the assumption used in the inference process and then to verify that they do not contradict each other and do not contradict with the conclusions.

**Definition 2.1[7]**: Let (D,W) be a default theory. For any set of formulas T, let $\Psi$(T) be the pair of the smallest sets of formulas (S',T') such that:

(1) W$\subseteq$S'$\subseteq$T';
(2) S'=Th(S') and T'=Th(T');
(3) For $\frac{\alpha:\beta}{\gamma} \in D$, if $\forall$0S' and $T \cup \{\beta\} \cup \{\gamma\}$ is consistent, then $\gamma \in$S' and$\beta \wedge \gamma \in$T'.

A pair (E,C) of sets of formulas is a *constrained extension* of (D,W) if and only if $\Psi$(E,C)=(E,C).

The set E is the *actual extension* of the default theory and C is the *set of constraints* (a consistent *context* for E), which keeps track of the justifications assumed to be true in the construction of E.

The *set of the generating defaults* for the constrained extension (E,C) is defined as follows:$GD_{\Delta}^{(E,C)} = \left\{ \frac{\alpha:\beta}{\gamma} | \alpha \in E \; and \; C \cup \{\beta, \gamma\} \not\vdash fals \right\}$.

This logical system satisfies the properties: semi-monotonicity, the existence of a constrained extension is guaranted, strong-regularity and commitment to assumption.

*Rational default logic* was proposed in [5] and is based on the idea that we cannot use in the construction of an extension defaults whose all justifications together are inconsistent with the extension.

**Definitions 2.2[5]**: Let (D,W) be a default theory, let X be a subset of the set D of defaults and let S be a set of formulas.

(1) We define $X_S = \left\{ \frac{\alpha}{\gamma} | \frac{\alpha:\beta_1,\ldots,\beta_n}{\gamma} \in X, \; S \cup \{\neg\beta_i\} \; inconsistent, \; 1 \leqslant i \leqslant n \right\}$ and Mon(X)=$\left\{ \frac{\Pr ereq(d)}{Conseq(d)} | d \in X \right\}$.
(2) A set X of defaults is *active* with respect to W and S if it satisfies the conditions:
   (i) Justif(X)=$\emptyset$ or Justif(X)$\cup$ S is consistent
   (ii) Prereq(X)$\subseteq Th^{X_S}$(W), where $Th^{X_S}$(W) is the deductive closure of W using classical inference rules and the monotonic rules from$X_S$.

(3) We denote by A(D,W,S) the set of all subsets of the defaults in D which are active with respect to W and S. $\emptyset \subseteq$ A(D,W,S). MA(D,W,S) is defined as the set of all maximal elements in A(D,W,S).

The set E of formulas is a *rational extension* for the theory (D,W) if E=$Th^{X_E}$(W), where X$\in$ MA(D,W,E), and X is the *set of generating defaults*.

The next definition is proposed by Schaub [8] and it is equivalent with the original definition of rational default logic.

**Definition 2.3[8]**: Let (D,W) be a default theory. For any set of formulas T, let $\Psi$(T) be the pair of the smallest sets of formulas (S',T') such that:

(1) W$\subseteq$S'$\subseteq$T';
(2) S'=Th(S') and T'=Th(T');
(3) For any $\frac{\alpha:\beta}{\gamma} \in D$, if $\forall$0S' and $T \cup \{\beta\}$ is consistent, then $\gamma \in$S' and$\beta \wedge \gamma \in$T'.

A pair (E,C) of sets of formulas is a *rational extension* of (D,W) if and only if $\Psi$(E,C)=(E,C). The set E is the actual extension of the default theory and C is the reasoning context.

The *set of the generating defaults* for the rational extension (E,C) is defined as follows: $GD_{\Delta}^{(E,C)} = \left\{ \frac{\alpha:\beta}{\gamma} | \alpha \in E \; si \; C \cup \{\beta\} \not\vdash fals \right\}$.

Rational default logic is a generalisation of constrained default logic that means: each rational extension of a default theory is also a constrained extension of the same theory. It can be easy observed that the sets of generating defaults for constrained extensions are active sets, not necessarily maximal active sets. Constrained default logic and rational default logic coincide on the class of semi-normal default theories. This logical system is strongly regular, does not guarantee the existence of extensions, is not semi-monotonic and does not commit to assumptions.

The following theorems provide characterisations for constrained, respective rational extensions of a default theory, using the set of generating defaults.

**Theorem 2.1[8]:** Let (D,W) be a default theory and let E and C be sets of formulas. Then (E,C) is a constrained extension of (D,W) if and only if E=Th(W$\cup$Conseq(D')) and C=Th(W$\cup$Justif(D')$\cup$Conseq(D')) for a maximal set D'$\subseteq$D such that D' is grounded in W and W$\cup$Justif(D')$\cup$Conseq(D') is consistent.

**Theorem 2.2[4]:** Let (D,W) be a default theory and let E and C be sets of formulas. Then (E,C) is a rational extension of (D,W) if and only if E=Th(W $\cup$ Conseq(D')) and C=Th(W $\cup$ Justif(D') $\cup$ Conseq(D')) for a maximal D' $\subseteq$ D such that D' is grounded in W and are satisfied the following conditions:

(i) W$\cup$Conseq(D')$\cup$Justif(D') is consistent
(ii) $\forall d \in$ D\D' we have: W$\cup$Conseq(D')$\cup\{\neg$Precond(d)$\}$ is consistent or W$\cup$Conseq(D')$\cup$Justif(D'$\cup${d}) is inconsistent.

The condition (ii) from the above theorem states that the set of generating defaults is maximal active with respect to W and E.

The two variants of the default logic presented in this section have as a common feature the fact that the inference process formalised by them is guided by a reasoning consistent context, which contains the beliefs and the underlying assumptions, used for deriving new nonmonotonic theorems.

## 3. Semantic characterization for constrained extensions and rational extensions of a default theory

In this section we will provide semantic characterization for constrained and rational extensions of a default theory. This operational approach is inspired from [3], the construction of an extension is viewed as a successive application of the applicable defaults.

In the following some notions from the semantic of first order logic will be used.

**Definition 3.1**:

(1) For each class of frames Λand a formula A we denote by Λ(A)={M|M∈ Λand $\mid =_M$A} the models of A, that means the set of all the frames from Λ in which the formula A is true.
(2) The frame M is a *model* for the set S of formulas, if and only if are satisfied: $\mid =_M$A, $\forall A \in S$. We will use the notation $\mid =_M$S.
(3) A *class* Λ of frames is *elementary* if and only if Λ is the class of all the models of the set S of formulas.

The particularity of these two variants of default logic that are used implicit assumptions to derive new explicit conclusions suggests that there is an explicit content (the set of beliefs) and an implicit content (assumptions) of the knowledge base. These two aspects must be correlated in the semantic characterisation of an extension.

Constrained and rational extensions are defined using a pair (E=actual extension, C=reasoning context). Thus it is naturally to have a pair $< \Lambda_1, \Lambda_2 >$ which characterises semantic these types of extensions as follows: $\Lambda_1$ is the class of all the models of the set E of beliefs, and $\Lambda_2$ is the class of all the models of the context C. We have $\Lambda_2 \subseteq \Lambda_1$ since E⊆C.

**Definition 3.2:** Let $\Lambda_1$ and $\Lambda_2$ be two classes of frames. The pair $< \Lambda_1, \Lambda_2 >$ is called a bi-*structure* if and only if $\Lambda_1$ and $\Lambda_2$are elementary classes and $\Lambda_2 \subseteq \Lambda_1$.

A bi-structure $< Lambda_1, \Lambda_2 >$ characterises the stage of the reasoning process as follows: $\Lambda_1$represents the set of all the models of a set of beliefs and $\Lambda_2$ represents the set of all the models of a set of formulas, which represent the reasoning context.

*reasoning context* = set of beliefs (non-monotonic theorems) + implicit assumptions (justifications of the used defaults)

**Definition 3.3**: Let $< \Lambda_1, \Lambda_2 >$ be a pair of frames-frames and $d = \frac{\alpha : \beta_1, \ldots, \beta_m}{\gamma}$ a default.

(1) The default d is *res-applicable with respect to* $< \Lambda_1, \Lambda_2 >$ if and only if: (i) $\mid =_M \alpha$, $\forall M \in \Lambda_1$ and (ii) $\exists M \in \Lambda_2$ *a.i.* $\mid =_M \beta \wedge \gamma$
(2) The default d is *rat-applicable with respect to* $< \Lambda_1, \Lambda_2 >$ if and only if: (i) $\mid =_M \alpha$, $\forall M \in \Lambda_1$ and (ii) $\exists M \in \Lambda_2$ *a.i.* $\mid =_M \beta$

The conditions of res-applicability and rat-applicability are the semantic counterpart of the applicability conditions from definitions 2.1 and 2.3. We can interpret these semantic conditions of applicability as follows:

- res-applicable with respect to $< \Lambda_1, \Lambda_2 >$ if and only if the prerequisite is believed and the justification together with the consequent are consistent with the reasoning context.
- rat-applicable with respect to $< \Lambda_1, \Lambda_2 >$ if and only if the prerequisite is believed and the justification is consistent with the reasoning context.

**Definition 3.4:** To a closed default $d = \frac{\alpha:\beta_1,\ldots,\beta_m}{\gamma}$ we assign a mapping $d^{res}$ from the set of the bi-structures into the set of bi-structures as follows:

$$d^R(<\Lambda_1,\Lambda_2>) = \begin{cases} <\Lambda_1(\gamma),\Lambda_2(\beta\wedge\gamma)> & \text{if } d \text{ is res-applicable wrt } <\Lambda_1,\Lambda_2> \\ <\Lambda_1,\Lambda_2> & \text{otherwise} \end{cases}$$

This mapping models a *cautious application* of the defaults which means that the commitment to assumptions for each applied default is guaranteed, and then inconsistencies after the application of an applicable default cannot be obtained.

**Definition 3.5:** To a closed default $d = \frac{\alpha:\beta_1,\ldots,\beta_m}{\gamma}$ we assign a mapping $d^{rat}$ from the set of frames-frames into the set of frames-frames as follows:

$$d^{rat}(<\Lambda_1,\Lambda_2>) = \begin{cases} <\Lambda_1(\gamma),\Lambda_2(\beta\wedge\gamma) & \text{if } <\Lambda_1,\Lambda_2> \text{ is a bi-structure and} \\ & d \text{ is rat-aplicable wrt } <\Lambda_1,\Lambda_2> \\ <\Lambda_1,\Lambda_2> & \text{if } <\Lambda_1,\Lambda_2> \text{ is a bi-structure and} \\ & d \text{ is not rat-aplicable wrt } <\Lambda_1,\Lambda_2> \\ <\emptyset,\emptyset> & \text{if } <\Lambda_1,\Lambda_2> \text{ is not a bi-structure} \end{cases}$$

The above definition models a step in the reasoning process, where the commitment to assumptions is not guaranteed. We say that we have a *hazardous application* of the defaults that means: the application of an applicable default can cause inconsistencies in the set of beliefs or in the reasoning context.

**Definition 3.6:** Let $<\Lambda_1,\Lambda_2>$ be a bi-structure and D a set of closed defaults.

(1) $<\Lambda_1,\Lambda_2>$ *is res-stable wrt D* if and only if $d^{res}(<\Lambda_1,\Lambda_2>)=<\Lambda_1,\Lambda_2>$, $\forall d \in D$.
(2) $<\Lambda_1,\Lambda_2>$ is *rat-stable wrt D* if and only if $d^{rat}(<\Lambda_1,\Lambda_2>)=<\Lambda_1,\Lambda_2>$, $\forall d \in D$.

A stable bi-structure characterises the end of the reasoning process in which all the applicable defaults were used.

**Definition 3.7:** Let $<\Lambda_1,\Lambda_2>$ be a pair of frame-frame and $<d_i>$ a sequence of closed defaults.

(1) We denote by $<d_i>^{res}(<\Lambda_1,\Lambda_2>)$ the bi-structure obtained as follows:

$$<d_i>^{rat}(<\Lambda_1,\Lambda_2>) = \begin{cases} <\Lambda_1,\Lambda_2> & \text{if } <d_i>=\emptyset \\ <\cap\Lambda_1^i,\cap\Lambda_2^i> & \text{else} \end{cases}$$

where $<\Lambda_1^0,\Lambda_2^0>=<\Lambda_1,\Lambda_2>$, and $<\Lambda_1^{i+1},\Lambda_2^{i+1}>= d_i^{res}(<\Lambda_1^i,\Lambda_2^i>)$ for i=1,2,...

(2) We denote $<d_i>^{rat}(<\Lambda_1,\Lambda_2>)$ the pair frames-frames obtained as follows:

$$<d_i>^{res}(<\Lambda_1,\Lambda_2>) = \begin{cases} <\Lambda_1,\Lambda_2> & \text{if } <d_i>=\emptyset \\ <\cup\Lambda_1^i,\cup\Lambda_2^i> & \text{else} \end{cases}$$

where $<\Lambda_1^0,\Lambda_2^0>=<\Lambda_1,\Lambda_2>$, and $<\Lambda_1^{i+1},\Lambda_2^{i+1}>= d_i^{rat}(<\Lambda_1^i,\Lambda_2^i>)$ for i=1,2,...

These definitions model a reasoning process that consists in a successive application of the elements from a sequence of defaults.

**Definition 3.8:** Let $< \Lambda_1,\Lambda_2 >$ be a bi-structure, let Z be an elementary class of frames and $< d_i >$ a sequence of closed defaults. The pair $< \Lambda_1,\Lambda_2 >$ is $< d_i >$-x-*accessible from Z* if and only if $< \Lambda_1,\Lambda_2 >=< d_i >^x(<$ Z,Z$>)$. $< \Lambda_1,\Lambda_2 >$ is x-*accessible* from Z *wrt D* if and only if there exists a sequence $< d_i >$ of defaults in D such that $< \Lambda_1,\Lambda_2 >$ is $< d_i >$-x-accessible from Z. x can be *res* or *rat*, and thus the notions of *res-accessibility* and *rat-accessibility* are defined.

Using the notions presented before, the following theorems provide semantic characterisations for constrained extensions, respective rational extensions.

**Theorem 3.1 (correctness and completeness):** Let (D,W) be a closed default theory and let Z be the class of all models of W. A class of frames $\Lambda_1$ is the class of all models of actual extension E and $\Lambda_2$is the class of all models of the reasoning context C (where (E,C) is a constrained extension of (D,W)) if and only if there exists a bi-structure $< \Lambda_1,\Lambda_2 >$ which satisfies:

(i) $< \Lambda_1,\Lambda_2 >$ is res-stable wrt D and (ii) $< \Lambda_1,\Lambda_2 >$ is res-accessible from Z wrt D.

**Proof:**

(**correctness**) Assume that (E,C) is a constrained extension for (D,W), then according to theorem 2.1 we have that:

E=Th(W∪Conseq(D')), C=Th(W∪Conseq(D')∪Justif(D')), where D' is grounded in W and the set W∪Conseq(D') ∪Justif(D') is consistent.

Let $\Lambda_1$={M| $\models_M W \cup Conseq(D')$} be the set of all models of the actual extension E and let $\Lambda_2$={M| $\models_M W \cup Conseq(D') \cup Justif(D')$} be the set of all models of the reasoning context C. We have then that $< \Lambda_1,\Lambda_2 >$ is a bi-structure and we have to verify that $< \Lambda_1,\Lambda_2 >$ satisfies conditions (i) and (ii) from the theorem:

- For (i) we have to prove that $\forall d \in D : d^{res}(< \Lambda_1,\Lambda_2 >)=< \Lambda_1,\Lambda_2 >$

There are two cases:

1. For $d = \frac{\alpha:\beta}{\gamma} \in D'$:

$\alpha \in$E, hence $\models_M \alpha$, $\forall M \in \Lambda_1$ **and**

$\gamma \in$E, C$\cup\{\beta \wedge \gamma\}$ is consistent, hence $\exists M \in \Lambda_2$ *such that* $\models_M \beta \wedge \gamma$

We have $\Lambda_1(\gamma) = \Lambda_1$ since $\gamma \in$E, and $\Lambda_2(\beta \wedge \gamma)¿ =\Lambda_2$ since$\beta \wedge \gamma \in$C.

The default d is res-applicable wrt $< \Lambda_1,\Lambda_2 >$ according to definition3.3 and $d^{res}(< \Lambda_1,\Lambda_2 >)= < \Lambda_1(\gamma),\Lambda_2(\beta \wedge \gamma) >= < \Lambda_1,\Lambda_2 >$

2. For $d = \frac{\alpha:\beta}{\gamma} \in D\backslash D'$:

$\alpha \notin$E, hence $\exists M \in \Lambda_1$ *such that* $\not\models_M \alpha$ **or**

C$\cup\{\beta \wedge \gamma\}$ is inconsistent, hence $\nexists M \in \Lambda_2$ *such that* $\models_M \beta \wedge \gamma$

According to the definition 3.3 the default d is not res-applicable wrt $< \Lambda_1,\Lambda_2 >$ and $d^{res}(< \Lambda_1,\Lambda_2 >)=< \Lambda_1,\Lambda_2 >$

Thus we have proved the res-*stability* of the *bi-structure* $< \Lambda_1,\Lambda_2 >$ wrt D.

- For (ii) we have to prove that there exists a sequence of defaults $< d_i >$ such that $< \Lambda_1,\Lambda_2 >=< d_i >^{res}(<$Z,Z$>)$, where Z={M| $\models_M$ W}.

The set D' is grounded in W, therefore exists an enumeration $\langle \delta_i \rangle_{i \in I}$ of its elements such that:

(1) $W \cup Conseq(\{\delta_0, \delta_1, \ldots, \delta_{i-1}\}) \mapsto \Pr econd(\delta_i), \forall i \in I = \{0, 1, 2, n\}$

We consider that this enumeration represents the sequence $< d_i >$, which provides the application order of the defaults for generating the constrained extension (E,C).

a) if $< d_i >= \emptyset$ then $< \Lambda_1, \Lambda_2 >=<$Z,Z$>$, the set of generating defaults D'=$\emptyset$ and the constrained extension (E,C), E=C=Th(W).
b) if $< d_i > \neq \emptyset$ we show by induction that:

(2) $< d_0, ..., d_k >^{res} (< Z, Z >) =< \Lambda_1^{k+1}, \Lambda_2^{k+1} >, k = 0, \ldots, n,$ where

$\Lambda_1^{k+1} =\{\text{M}| \quad| =_M W \cup Conseq(\{d_0, ..., d_k\})\}$ and
$\Lambda_2^{k+1} =\{\text{M}| \quad| =_M W \cup Conseq(\{d_0, ..., d_k\}) \cup Justif(\{d_0, ..., d_k\})\}$

**Base**: k=0

From (1) with i=0 we have: W$\mapsto$ $\Pr econd(d_0)$ which implies W$|$ = $\Pr econd(d_0)$, therefore $\forall M \in Z \quad| =_M \Pr econd(d_0)$. The set W$\cup$Conseq($d_0$) $\cup$Justif($d_0$) is consistent because is a subset of the consistent set W$\cup$Conseq(D')$\cup$Justif(D')).

Hence $\exists M \in Z$ *such that* $| =_M Justif(d_0) \wedge Conseq(d_0)$.

Thus are satisfied the res-applicability conditions for the default $d_0$ wrt <Z,Z>.

$d_0^{res}$(<Z,Z>)= $< \Lambda_1^1$=Z(Conseq($d_0$),$\Lambda_2^1$=Z(Justif($d_0$)$\wedge$Conseq($d_0$)) >.

**Step:** Let us assume that the relation (2) is true for k and we will prove that it is true for k+1.

From (1) with i=k we have: W$\cup$Conseq($\{d_0, \ldots, d_k\}$)$\mapsto$ $\Pr econd(d_{k+1})$ which implies W$\cup$Conseq($\{d_0, \ldots, d_k\}$)$|$ = $\Pr econd(d_{k+1})$, and then $\forall M \in \Lambda_1^{k+1} \quad| =_M \Pr econd(d_{k+1})$.

W$\cup$Conseq($\{d_0, ..., d_k, d_{k+1}\}$)$\cup$Justif($\{d_0, \ldots, d_k, d_{k+1}\}$) is a consistent set because is a subset of the consistent set W$\cup$Conseq(D')$\cup$Justif(D').

Hence $\exists M \in \Lambda_2^{k+1}$ *such that* $| =_M Justif(d_{k+1}) \wedge Conseq(d_{k+1})$

Thus are satisfied the res-applicability conditions for the default $d_{k+1}$ wrt $< \Lambda_1^{k+1}$,$\Lambda_2^{k+1} >$.

$< d_0, ..., d_{k+1} >^{res}$(<Z,Z>) = $d_{k+1}^{res}$(¡$\Lambda_1^{k+1}$,$\Lambda_2^{k+1} >$)= $< \Lambda_1^{k+1}$ (Conseq( $d_{k+1}$)), $\Lambda_2^{k+1}$ (Conseq($d_{k+1}$)$\wedge$ Justif $d_{k+1}$)) > = $< \Lambda_1^{k+2}$, $\Lambda_2^{k+2} >$ and thus the relation (2) is satisfied for k+1.

For i=n we have: $< d_0, \ldots, d_n >^{res}$ (<Z,Z>) = $< \Lambda_1^{n+1}$,$\Lambda_2^{n+1} >$ = $<\{\text{M}|| =_M W \cup Conseq(\{d_0, \ldots, d_n\})\}$, $\{\text{M}|| =_M W \cup Conseq(\{d_0, \ldots, d_n\}) \cup Justif(\{d_0, \ldots, d_n\})\}¿ = < \Lambda_1, \Lambda_2 >$ since D'=$\{d_0, ..., d_n\}$.

Thus was proved the *res-accessibility* of the *bi-structure* $< \Lambda_1, \Lambda_2 >$ from Z wrt D.

(**completeness**) We assume that $< \Lambda_1, \Lambda_2 >$ is a bi-structure which satisfies the conditions (i) and (ii). $\Lambda_1$ is the set of all the models of the set E of formulas and $\Lambda_2$is the set of all the models of the set C of formulas. The relation $\Lambda_1 \supseteq \Lambda_2$ implies E$\subseteq$C.

We have to prove that (E,C) is a constrained extension for the default theory (D,W).

According to the condition (ii) there exists a sequence of defaults $< d_i >= d_0, ..., d_n$in D such that $< d_0, ..., d_n >^{res}$(<Z,Z>)= $< \Lambda_1,\Lambda_2 >$, where Z={M| $\mid =_M$W}

**If** $< d_i >= \emptyset$ **then** $< \Lambda_1,\Lambda_2 >$ = <Z,Z> .

Since <Z,Z> is a bi-structure res-stable wrt D we have that (E=Th(W),C=Th(W)) is a constrained extension for (D,W). The set of the generating defaults is $\emptyset$.

**If** $< d_i >\neq \emptyset$ **then**:

Following step by step the application of the defaults in the sequence we observe that:

(a0) $\mid =_M$Precond($d_0$), $\forall M \in Z = \Lambda_1^0$ and thus W$\mapsto$ Pr$econd(d_0)$

(b0) $d_0^{res}$(<Z,Z>)=< $\Lambda_1^1$ = Z(Conseq($d_0$), $\Lambda_2^1$ = Z(Justif($d_0$)$\wedge$Conseq($d_0$)) > = <{M| $\mid =_M W \cup Conseq(d_0)$}, {M| $\mid =_M W \cup Conseq(d_0) \cup Justif(d_0)$}¿.

$< \Lambda_1^1,\Lambda_2^1 >$ is a bi-structure which implies that W$\cup$Conseq($d_0$)$\cup$Justif($d_0$) is a consistent set.

Using the notations:

$\Lambda_1^{k+1}$ ={M| $\mid =_M W \cup Conseq(\{d_0, ..., d_k\})$} and

$\Lambda_2^{k+1}$ ={M| $\mid =_M W \cup Conseq(\{d_0, ..., d_k\}) \cup Justif(\{d_0, ..., d_k\})$}

we can easily prove by induction that for k=1,2,. . . , n are satisfied the relations:

(ak) $\mid =_M$Precond($d_k$), $\forall M \in \Lambda_1^k$, hence W$\cup$Conseq($\{d_0, ..., d_k\}$)$\mapsto$ Pr$econd(d_{k+1})$

(bk) $< d_0, ..., d_k >^{res}$(<Z,Z>)=< $\Lambda_1^{k+1}$, $\Lambda_2^{k+1}$ > is a bi-structure.

From (a0) + (a1) +. . . +(an) we have that the set D'=$\{d_0, ..., d_n\}$ is grounded in W.

For k=n we have: $< d_0, ..., d_n >^{res}$(<Z,Z>)= $< \Lambda_1,\Lambda_2 >$ = <{M| $\mid =_M$ W $\cup$Conseq(D')}, {M| $\mid =_M$ W $\cup$Conseq(D') $\cup$Justif(D')}¿

$< \Lambda_1,\Lambda_2 >$ is a bi-structure and thus W$\cup$Conseq(D')$\cup$Justif(D') is a consistent set.

The res-stability condition for $< \Lambda_1,\Lambda_2 >$ means that $d^{res}(< \Lambda_1,\Lambda_2 >)=< \Lambda_1,\Lambda_2 >$, for every d in D :

If d$\in$D' then d is res-applicable wrt $< \Lambda_1,\Lambda_2 >$ but it was applied already.

If d$\in$D\D' then we have two cases (i) or (ii):

(i) $d$ is res-applicable wrt $< \Lambda_1,\Lambda_2 >$ but applying it we can neither obtain new nonmonotonic theorems nor modify the context. We add to D' all the defaults d with this property: D'=D'$\cup${d} and D' remains grounded in W.

(ii) $d$ is not res-applicable wrt $< \Lambda_1,\Lambda_2 >$ due to the following:
$\exists M \in \Lambda_1$ $a.i.$ $\mid \neq_M$ Pr$econd(d)$,thus D'$\cup${d} is not grounded in W **or**
$\nexists M \in \Lambda_2$ $a.i$ $\mid =_M Justif(d) \wedge Conseq(d)$, thus
W$\cup$Conseq(D'$\cup${d})$\cup$Justif(D'$\cup${d}) is an inconsistent set.

All the defaults in D\D' are not res-applicable wrt $< \Lambda_1,\Lambda_2 >$ and thus is guaranteed the maximality of D' such that D' is grounded in W and W$\cup$Conseq(D')$\cup$Justif(D') is consistent.

The maximal sets E, C of formulas with the property that $\Lambda_1,\Lambda_2$ are the sets of all their models respectively, have the form: E=Th(W$\cup$Conseq(D'), C=Th(W$\cup$Conseq(D')$\cup$Justif(D')) and thus (E,C) is a constrained extension according to the theorem 2.1.

**Theorem 3.2 (correctness and completeness):** Let (D,W) be a closed default theory and Z be the class of all models of W. A class of frames $\Lambda_1$ is the class of all the models of the actual extension E and $\Lambda_2$is the class of all the models of the reasoning context C (where (E,C) is a rational extension of the theory (D,W)) if and only if there exists a bi-structure $< \Lambda_1,\Lambda_2 >$ which satisfies the following conditions:

(i) $< \Lambda_1,\Lambda_2 >$ is rat-stable wrt D and (ii) $< \Lambda_1,\Lambda_2 >$ is rat-accessible from Z wrt D.

**Proof:** The proof of this theorem is similar to the above proof. Theorem 2.2 can be used for the characterization of rational extensions.

## 4. Graphical interpretation of the semantic characterization of constrained and rational extensions

To each default theory (D,W) we can associate a transition network. The nodes of this network contain pairs frames-frames and the arcs are labelled with defaults from the set D. If a node contains a bi-structure is called *viable*, otherwise is called *contradictory.* A *leaf node* is a node whose outbound arcs loop back.

The network is specified as follows:

(1) The set of nodes is the smallest set which satisfies the conditions:
- $<$ Z, Z ¿ is the root node, where Z={M| $\quad| =_M$W}.
- if **n** is a viable node and d∈D, then $d^{res}$(n) (respective $d^{rat}$(n)) is a node of the network.

(2) From each viable node **n** and for each d∈D, there is an arc label by d which leads to the node $d^{res}$(n) (respective $d^{rat}$(n))

We can give a graphical interpretation for the theorems 3.1 and 3.2.

Let (D,W) be a default theory and the associated transition network built using $d^{res}$(respective $d^{rat}$). Every viable node characterises a *constrained extension* (respective *rational extension)* for the default theory (D,W) as follows:

$< \Lambda_1,\Lambda_2 >$ is a stable bi-structure contained in a leaf node, where:
$< \Lambda_1,\Lambda_2 >=< d_i >^{res}(<$ Z,Z$>)$ (respective $< \Lambda_1,\Lambda_2 >=< d_i >^{rat}(<$ Z,Z$>)$),
$< d_i >=d_1,\ldots,d_k$ , and Z={M| $\quad| =_M$W}.
**if and only if**

$\Lambda_1$is the class of all models of the set E=Th$(W \cup \bigcup_{i=1}^{k} Conseq(d_i))$ and $\Lambda_2$ is the class of all the models of the set of formulas C=Th$(W \cup \bigcup_{i=1}^{k} Conseq(d_i) \cup \bigcup_{i=1}^{k} Justif(d_i))$, that means:

$$\Lambda_1 = \{M|| =_M W \cup \bigcup_{i=1}^{k} Conseq(d_i)\},$$

$$\Lambda_2 = \{M|| =_M W \cup \bigcup_{i=1}^{k} Conseq(d_i) \cup \bigcup_{i=1}^{k} Justif(d_i)\}$$

and $\{d_1, \ldots, d_k\}$ is the set of generating defaults for the constrained extension (E,C) (respectively for the rational extension (E,C)).

**Example 4.1**: The default theory $(\{d1 = \frac{:B}{C}, d2 = \frac{:\neg B}{D}, d3 = \frac{:\neg C \wedge \neg D}{E}\}, \emptyset)$ has three constrained extensions corresponding to the leaf nodes of the transition network from the fig1.

(E1=Th({C}),C1=Th({C∧B})) with {d1} as the set of generating defaults.

(E2=Th({D}),C2=Th({D∧¬B})) with {d2} as the set of generating defaults.

(E3=Th({E}),C3=Th({E∧¬C∧¬D})) with {d3} as the set of generating defaults.

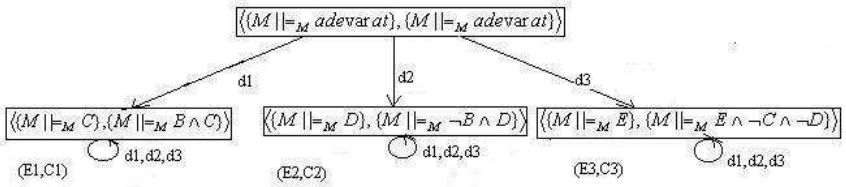


FIGURE 1.

Fig2 contains the transition network that characterizes the rational extensions. The same default theory has only two rational extensions: (E1,C1) and (E2,C2).

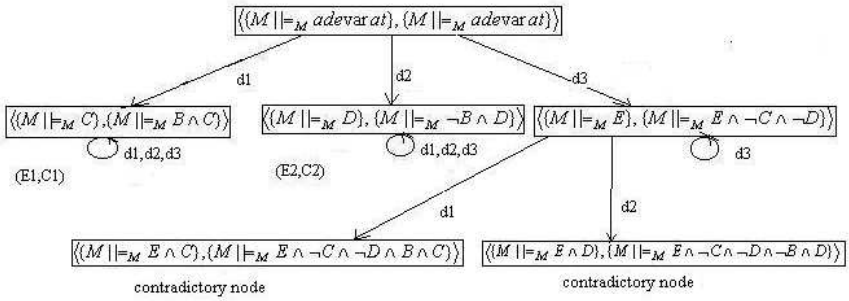


FIGURE 2.

According to the original definition of rational default logic we have that:

- {d1}∈ MA(D,W,E1), where E1=Th(W∪Conseq({d1})).
- {d2}∈ MA(D,W,E2), where E2=Th(W∪Conseq({d2})).
- {d1,d3},{d2,d3}∈ MA(D,W,S), where S=Th(W∪Conseq({d3})). Therefore the set {d3} is not maximal active with respect to W and S; hence {d3} cannot be a set of generating defaults for a rational extension.

The existence of the contradictory node in fig2 illustrates the semantical failure of semi-monotonicity, and thus the failure of commitment to assumption for rational default logic.

## 5. Conclusions

The semantic characterization of constrained and rational extensions proposed in this paper together with the semantic characterization of classical and justified extensions from [3] can be viewed as a uniform approach of semantics which permits a comparative study of the formal properties of these variants of default logic. We can observe similarities between classical and rational default logics, respective justified and constrained default logic as follows:

***justified - constrained default logics:***

- cautious application of the defaults
- a transition network which models the reasoning process in justified or constrained default logic does not have contradictory nodes
- these variants of default logic satisfy the semi-monotonicity property which permits to successively apply one default after another with no risk of destroying any previous partial extension
- a default theory has always justified and constrained extensions.

***classical - rational default logics***

- hazardous application of the defaults
- there is the possibility to obtain contradictory nodes, which means: a default that satisfies the applicability condition, after its application can cause inconsistencies in the set of beliefs or in the reasoning context
- the failure of the semi-monotonicity property, and thus a classical and a rational extension can not be generate iteratively
- the existence of classical and rational extensions is not guaranteed: there are transition networks having only contradictory nodes as final nodes
- does not commit to assumptions

Constrained default logic satisfies the property of commitment to assumptions due to the fact that the res-applicability condition is cautious and the reasoning context must be consistent.

Rational and constrained default logics are strong-regular because the applicability conditions for defaults require the reasoning context to be consistent.

## References

[1] D.W. Etherington: A semantics for default logic. Proceedings of IJCAI 1987, pp. 495-498.
[2] W. Lukaszewicz: Two results on default logic. Proceedings of the IJCAI, 1985.
[3] W. Lukaszewicz: Non-monotonic reasoning. Ellis Horwood Limited, 1990.
[4] M.Lupea: A comparative study of versions of default logic, presented on the Ph.D. program, November 2001, Faculty of Mathematics and Computer Science, "Babes-Bolyai" University of Cluj-Napoca, Romania.
[5] A.Mikitiuk, M.Truszczynski: Rational default logic and disjunctive logic programming, în A. Nerode, L.Pereira, Logic programming and non-monotonic reasoning, MIT Press, 1993, pp. 283-299.
[6] R.Reiter: A logic for default reasoning, Journal of Artificial Intelligence, 13, 1980, pp. 81-132.
[7] T.H.Schaub: Considerations on default logics. Ph.D. Thesis, Technischen Hochschule Darmstadt, Germany, 1992.
[8] T.H. Schaub. The Automation of Reasoning with Incomplete Information. Springer-Verlag Berlin, 1997.

Department of Computer Science, Babeş-Bolyai University, 1 M. Kogălniceanu St., RO-3400 Cluj-Napoca, Romania
*E-mail address*: mlupea@cs.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLVI**, Number 2, 2001

# ON ROMANIAN ARTICLE SEMANTICS

DANA AVRAM

Abstract. In this paper a way to represent Romanian article semantic is proposed. When trying to represent sentence semantics by using first order predicate calculus, the articles usually became the mathematical operators $\exists$ and $\forall$. We define a more powerful operator, called DET, that encloses the significance of $\exists$ and $\forall$. Rules that may be used with DET in a FOPC are considered. This proposed representation is also appropriate for other determiners class.

## 1. Introduction

The kinds of grammars that we are familiar with do not model the reasoning process of the brain. They are descriptions of the natural language structure to a certain degree of precision. So the course of development of the brain cannot really be explained by the formal properties of the descriptive apparatus. Similar questions can be raised for every piece of knowledge, specific and general, that is embodied in the mental grammar. The set of things that we characterize as prior knowledge are part of the language faculty, or universal grammar [2]. Everything else is learned. The question, now, is how?

Deduction systems are one of the most used expert systems. They have a set of statements (dates) and a set of deduction rules for constructing new statements. They are built on mathematical logic operations whose fundamental rules are very well stated. This very rigid but rigorous system will be exploited in the next paragraphs.

## 2. Articles in Romanian Language

Syntactically speaking, the article appears near a noun [5]. They are definite, indefinite, or bare. The indefinite form usually indicates that the speaker has no information, or a reduced degree of information, on the object it stands by. The definite article indicates a higher degree of information. The bare article is an indefinite article, but its presence is not required by the syntactic rules. The singular and plural form, the definitiveness or indefinitiveness article forms have a

2000 *Mathematics Subject Classification.* 68T50.

1998 *CR Categories and Descriptors.* I.2.7[**Computing Methodologies**]: Artificial Intelligence – *Natural Language Processing*.

supplementary numeric role. We will concentrate on the second problem and will construct a model that captures this meaning in a simple and explicit way [4].

From the viewpoint of a syntactic form, the article may exist by himself as a separate word, standing above the noun, or it may be part of the noun it determines. The forms of the Romanian article are presented in Table 1.

TABLE 1. The Romanian article syntactic forms

| | **Article** | **Syntactic form** | **Number/Use** | **Example** |
|---|---|---|---|---|
| 1. | *-l,-le,-a -lui, -e, -i* | part of the noun | singular/definite reference | *cainele [the dog]* |
| 2. | *un, o, unei, unui* | separate word | singular/indefinite reference | *un caine [a dog]* |
| 3. | *(toti) -i,(toate)-le-lor* | separate word (optional) and part of the noun | plural/definite reference | *(toti) cainii [(all) dogs]* |
| 4. | *(niste) -le, -lor* | separate word (optional) and part of the noun | plural/indefinite reference | *niste caini [some dogs]* |
| 5. | BARE | | bare singular NP (mass term) | *caine [dog]* |
| 6. | BARE | | bare plural NP (generics) | *caini [dogs]* |

## 3. Representation using FOPC

Let $L = (\Sigma, F, A, R)$ be the first order predicate logic (FOPC):

$$\Sigma = V \cup C \cup (\cup F_j) \cup (\cup P_j) \cup \{\forall, \exists, \neg, \wedge, \vee, \rightarrow, (,)\},$$

where $V$ represents a set of symbols called variables, $C$ represents a set of symbols called constants, $F_j$ represents a set of function symbols with $j$ parameters, $P_j$ represents a set of function symbols with $j$ parameters, $P_j$ represents a set of predicate symbols with $j$ parameters, the set $\{\forall, \exists, \neg, \wedge, \vee, \rightarrow, (,)\}$ represents the logic operators, and $\exists$ and $\forall$ represent FOPC quantifiers.

One important class of semantic constructors is quantifiers class [3]. In the first order predicate calculus [8], the two quantifiers, $\forall$ and $\exists$, encode the articles' meaning.

Let us see how we can represent some sentences using the FOPC defined earlier.

The singular article, definite or indefinite (Table 1, lines 1 and 2), says that $\exists$ the material object that corresponds to the given noun.

The definite plural article (Table 1, line 3) has the meaning of $\forall$.

The indefinite plural article (Table 1, line 4) indicates the existence ($\forall$) of one or more objects of the type that corresponds to the given noun. Few more quantifiers are necessary.

The bare (missing) article (Table 1, lines 5 and 6) is usually interpreted as indefinite article.

TABLE 2. Examples of sentences representation using FOPC

| | **Romanian** | **English translation** | **Representation of Romanian sentence** | **The same sentence in English** |
|---|---|---|---|---|
| 1. | *Un caine latra* | *[A dog barks]* | $(\exists x$: (CAINE$(x))$ $\rightarrow$ LATRA$(x))$ | $(\exists x$: (DOG$(x))$ $\rightarrow$ BARK$(x))$ |
| 2. | *Toti cainii latra* | *[All dogs bark]* | $(\forall x$: (CAINE$(x))$ $\rightarrow$ LATRA$(x))$ | $(\forall x$: (DOG$(x))$ $\rightarrow$ BARK$(x))$ |
| 3. | *Niste caini latra* | *[Some dogs bark]* | $(\exists x$: (CAINE$(x))$ $\rightarrow$ LATRA$(x))$ | $(\exists x$: (DOG$(x))$ $\rightarrow$ BARK$(x))$ |

Looking at Table 2 we may remark that the sentence *Un caine latra* / [*A dog barks*] is represented in the same way as the sentence *Niste caini latra* / [*Some dogs bark*]. It is easy to notice that this representation looses a part of the natural language semantics. The problem is that any natural language contains a much larger range of quantifiers than the two from FOPC. As an example for the higher complexity of natural language quantifiers, let us consider the following FOPC formula

$$\forall x : P(x)$$

This formula is true if and only if $P(x)$ is true for every possible object in the domain.

Such statements are rare in natural language. We will rather say [*most dogs bark*] (and in this case, this is not an article domain, but a Romanian adverb) or [*some people laugh*], which requires constructs that are often called generalized quantifiers. These quantifiers are used in statements of the general form [1, 6]:

(quantifier variable: restriction proposition $\rightarrow$ body-proposition)

For example:

([NISTE]$(x)$:(CAINE$(x)) \rightarrow$ LATRA$(x))$
(SOME$(x)$:DOG$(x) \rightarrow$ BARK$(x))$

This roughly captures the meaning of the sentence: *If there are some things that are also dogs, then they are barking things.*

Or:

([CEI MAI MULTI]($x$):(CAINE($x$)) → LATRA(x))
(MOST($x$):DOG($x$) → BARK($x$))

This means that: *Most dogs are barking things.*

## 4. Quantifiers with extended functionality

A construct to handle plural forms, as in the phrase *two dogs bark* must to be introduced. This indicates not *a dog*, but *two*. It can easily be seen that the article has also a numeric meaning. Let us consider the general form:

DET[*variable*, *name*, *number*]

where *variable* is the variable inherited from FOPC, *name* is the name of the determinant, and *number* is the number of objects indicated by the noun, or the percent value (of all the possible objects in discussion) only indicators of number (as *all, some*) are specified. *Toti* [*all*] refer to 100%, some will be convenient for 25% (less than 50%). Tabel 3 shows the Romanian articles representations using the general form DET described above.

Table 3. Articles representations using DET

| | **Article** | **Representation** |
|---|---|---|
| 1. | *-l,-le,-a-lui, -e, -i* | DET[x, **article**, 1] |
| 2. | *un, o, unei, unui* | DET[x, **article**, 1] |
| 3. | *(toti) -i,(toate)-le-lor* | DET[x, **article**, 100%] |
| 4. | *Niste -le, -lor* | DET[x, **article**, 25%] |
| 5. | BARE | The same representation as the indefinite form and the same number |

For example, the previous expression

([NISTE]($x$):(CAINE($x$)) → LATRA($x$))

becomes

(DET [$x$, **niste**, 25%]:(CAINE($x$)) → LATRA($x$))

and the expression

([UN]($x$):(CAINE($x$)) → LATRA($x$))

becomes

(DET[$x$, **un**, 1]:(CAINE($x$)) → LATRA($x$))

For the adverbial expression *cei mai multi* [*most*] 75% will be convenient for (more than 50%).

For example, the expression:

([CEI MAI MULTI]($x$):(CAINE($x$)) → LATRA($x$))

becomes:

(DET [$x$, **cei mai multi**, 75%]:(CAINE($x$)) → LATRA($x$))

Numeral determiners make no assumption about the whole class of the object. Their number will appear on the third position on DET argument, as in the next example.

The expression:

([DOI]($x$):(CAINE($x$)) → LATRA($x$))

becomes:

(DET [$x$, **doi**, 2]:(CAINE($x$)) → LATRA($x$))

## 5. Rules for DET

To allow quantifiers, variables are introduced as in first order logic but with an important difference. in first order logic a variable only retains its significance within the scope of quantifier. Thus two instances of the same variable $x$ occurring in two different formulas – say in the formulas $\exists x P(x)$ and $\exists x Q(x)$ are treated as completely different variables with no relation to each other. Natural languages display a different behavior. For instance consider that two persons say the following two true sentences: *A dog barks* and *Three dogs bark*. The first sentence introduces a new object to the discussion namely *a dog*. You might think to treat the meaning of this sentence along the lines of the existential quantifier in logic. But the problem is that the dog number introduced existentially in the first sentence is completed by to the number *three* in the second sentence. Variables appear to continue their existence after being introduced and the associate determiners usually change by unification [1]. In FOPC they are combined using the logical operators $\{\neg, \wedge, \vee, \rightarrow, (, )\}$. Here $\rightarrow$ can be obtained from $\vee$ and $\neg$, and the parens (, ) specify the order. So we have to consider rules of combining DET with $\rightarrow, \wedge, \vee$.

We suppose that all the variables refer to the same variable univers (unique and known) which will be called the contextual universe.

Rules for $\neg$ are different when the number is percent and when it has a concrete value. As it will be seen, there are cases when taking a decision is improper.

**Rule no 1:** (Percent case)

$\neg$((DET [$x$, **det1**, p1]: OBJECT($x$)) = (DET [$x$, **det1**, 100% -$p$1]: $\neg$ OBJECT($x$))

**Rule no 2:** (Numeric case)

Suppose that we know the total number objects in the contextual universe.

if **tot** = total_nr_of_objects_that_determiner_determines is known then

$\neg$ (DET [$x$, **det1**, $p$1] : OBJECT($x$))= (DET [$x$, **det1**, **tot** -$p$1] : $\neg$ OBJECT($x$))

else

$$\neg\ (\text{DET}\ [x,\ \textbf{det1},\ p1]\ :\ \text{OBJECT}(x)) = \text{undefined}$$

The rule for $\wedge$ depends on maximal values of numeric argument. Suppose that $p1 > p2$.

**Rule no 3:**

$$((\text{DET}\ [x,\ \textbf{det1},\ p1]\ :\ \text{OBJECT}(x)) \wedge (\text{DET}\ [x,\ \textbf{det2},\ p2]\ :\ \text{OBJECT}(x))) = (\text{DET}\ [x,\ \textbf{det1},\ p1]\ :\ \text{OBJECT}(x))\ (\text{if}\ p1 > p2)$$

Rule for $\vee$ depends on numeric argument as in $\wedge$ case.

**Rule no 4:**

$$((\text{DET}\ [x,\ \textbf{det1},\ p1]\ :\ \text{OBJECT}(x)) \vee (\text{DET}\ [x,\ \textbf{det2},\ p2]\ :\ \text{OBJECT}(x))) = (\text{DET}\ [x,\ \textbf{det2},\ p2]\ :\ \text{OBJECT}(x))\ (\text{if}\ p1 > p2)$$

There are mathematical rules that link with $\exists$ and $\forall$. One of the simplest mathematical rules [8] says that $\forall$ implies $\exists$.

if

$(\forall x\text{: OBJECT}(x))$

then

$(\exists x\text{: OBJECT}(x))$

That means that: if *any x is object* is true, then *an x is object* is true, too. In the DET case that is:

**Rule no 5:**

if

(DET $[x,\ \textbf{det1},\ p1]$ : OBJECT $(x)$) and $p2 < p1$ (percent or numeric)

then

(DET $[x,\ \textbf{det1},\ p2]$ : OBJECT $(x)$).

This means that if there are $p1$ objects $x$ and $p2$ is such that $p2 < p1$, then there are also $p2$ objects $x$ (in the given contextual universe).

We saw that problems that appear in the percent case are solved if we know the total number of the objects in the contextual universe. This is so because in this case we can transform the percent value into a (real) numeric one, as in the rule that follows.

**Rule no 6:**

Suppose that **tot** is the total number of the objects OBJECT in the contextual universe and $p1$ is a percent value. Then the math says that: $p2 = p1/100 \times \textbf{tot}$, and $p2$ is a numeric value, i. e.:

(DET $[x,\ \textbf{det1},\ p1]$ : OBJECT $(x)$) = (DET $[x,\ \textbf{det1},\ p1/100 \times \textbf{tot}]$ : OBJECT $(x)$).

Taxonomies are valuable resources in Natural Language Processing and Artificial Intelligence. They consist of hypernym (generalization) and hyponym (specialization) relations between concepts [7]. The most known example of such an organization is WordNet – a lexical database organized as a general terminological

system that contains semantic classes organized hierarchical is a classical example. There are also other knowledge bases as a partially structured knowledge, as domain specific terminological systems. Those structures may be easily used, if available, to improve deduction rules into determiners domain.

Let us consider the case of hierarchical system generated by ISA arcs. Suppose that OBJECT1 ISA OBJECT2, like a DALMATIAN is a DOG.

**Rule no 7:**

if

(DET [$x$, **det1**, $p1$] : DALMATIAN ($x$)) and $p1$ is numeric

then

(DET [$x$, **det1**, $p1$] : DOG ($x$))

This deduction rule works as follows: if there are five Dalmatians that bite, then there are also (at least) five dogs that bite.

This rule does not work in the percent case. We cannot say that if there are 50% Dalmatians that bite, then there are also 50% dogs that bite, nor that if there are 50% dogs that bite, then there are also 50% Dalmatians that bite.

## 6. Further research

Examples regarding the article case have been discussed. The issues approached in this paper may be developed even for numerals and other adverbs (with determiner role). As we have seen, the article semantics is much the same as of the other parts of speech with determiner role.

Only the case of a unique universe has been considered. But in the real world, each speaker states truths about his own known, time changing universe which may be different from the others. There is always a possibility that the statement be not (exactly) true if reported to the general universe. The classification scheme, structured according to the state of current human knowledge is, also, not perfect. Sometimes, the hierarchy is not so well done, there are exceptions that must be handled. One solution would be to introduce a special parameter to DET argument list in order to handle those special cases.

Noun phrases serve many different language functions, and it is important to distinguish these functions when considering scoping issues. There are at least three major classes to consider. Those involving definite reference indicate that the listener should in principle be able to identify the object or set. Definite reference occurs, for example, with determiners as *the* as in *the dog* (an individual) or *the fat men* (a specific set). In any natural settings there will obviously be many dogs in the world, so the use of the context to identify the correct one is crucial for understanding the sentence. Identification is a problem of anaphora resolution, and has been widely discussed in the literature.

## References

[1] **Allen, J.**, Natural Language Understanding, *The Benjamin Cummings Publishing Company, New York*, 1995.

[2] **Culicover, P.W.**, Language acquisition and the architecture of the language faculty, *Proceedings of the Berkeley Formal Grammar Conference Workshop, The University of California, Berkeley, CSLI Publications, http://www-csli.stanford.edu/publications/*, 2000.

[3] **Gal, A., Lapalme, G., Saint-Dizier, P., Somers, H.**, Prolog for Natural Language Analysis, *John Wiley, London*, 1991.

[4] **Graur, Al.** et. al., Romanian Language Grammar, *Romanian Academy Publishing House*, 1966.

[5] **Jurafsky, D., Martin, J.M.**, Speech and Language Processing, *Prentice Hall, Inc., University of Colorado, New Jersy, USA*, 2000.

[6] **Onet, A.**, A module-based application for te semantic representation of natural language sentences, **Proceedings of Eurolan 2001, Iasi, Romania**, 2001.

[7] **Stevenson, M.**, Enriching Noun Taxonomies with Thesaural Information, *Proceedings of NAACL 2001, Carnegie Mellon University Pittsburgh, PA, USA*, 2001.

[8] **Tatar, D.**, Artificial Inteligence: Automate Demonstration, Natural Language Processing, *Editura Albastră*, 2001.

Babeş-Bolyai University, Department of Computer Science, Cluj-Napoca, Romania
*E-mail address*: davram@cs.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLVI**, Number 2, 2001

# ACTIVECASE - TOOL FOR DESIGN OF CONCURRENT OBJECT-ORIENTED APPLICATIONS

DAN MIRCEA SUCIU

Abstract. Object-oriented concurrent programming is a methodology that seems to satisfy nowadays requirements for complex application development. Issues like inheritance anomalies or developing of object models that integrate in a natural way concurrent programming elements with object-oriented concepts was intensely analyzed in literature.

Construction of a consistent modeling mechanism that ameliorates the inheritance anomalies as much as possible represents the main goal of our research work ([13], [14]). This paper presents the implementation of this modeling mechanism into a CASE tool for analysis and design of concurrent object-oriented applications. Developing specific scalable statecharts for behavior modeling of active objects and automatic code generation are subsequent issues that are attend to validate the executability of our mechanism.

**Key words**: CASE tools, object-oriented concurrent programming, reactive systems, statecharts.

## 1. Introduction

CASE (*Computer Aided Software Engineering*) tools are software products able to support medium or large application development. This support is realised by automating some of the activities made in an analysis and design method. If we agree that one of the main goals of an analysis and design method is code generation and that we should obtain automatically a high rate of application code, it is obvious that an efficient use of a method cannot be made without an associated CASE tool.

Typically, the translation of a complex analysis/design model into a programming language takes a long period. A model is called *executable* if this translation can be made automatically. The automatization of the translation process allows running a prototype of an application immediately after building its model.

The executability is an important feature of *scalable statecharts* [13], allowing the automatization of active objects implementation based on their behavioral

2000 *Mathematics Subject Classification.* 68N30.

1998 *CR Categories and Descriptors.* D.2.3 [**Software**] : Software Engineering – *Coding Tools and Techniques*; D.2.7 [**Software**] : Software Engineering – *Distribution, Maintenance and Enhancements* .

models. Furthermore, the executability offers support for simulation, testing and debugging of active object execution at the same level of abstraction like the built model.

The paper describes the architectural and functional features of a CASE tool designed for modeling, developing and simulation of concurrent object oriented applications. This tool, called ActiveCASE, is complete original and supports active objects behaviour modeling through scalable statecharts formalism as described in [13] and [14]. In addition, ActiveCASE allows concurrent class structure specification, active object behavior modeling and source code generation.

Section 2 presents the meta-model of class diagrams and scalable statecharts implemented in ActiveCASE.

A detailed description of tool functional features is shown in section 3.

Section 4 validates the modeling capacity and executability of scalable statecharts describing the development and modeling process of an application for traffic control on a rectangular track. This sample exploits all scalable statecharts features described in [13] and [14].

## 2. The meta-model of scalable statecharts

In this section is presented in detail a static meta-model of scalable statecharts. This meta-model is used in scalable statecharts implementation in ActiveCASE (figure 1).

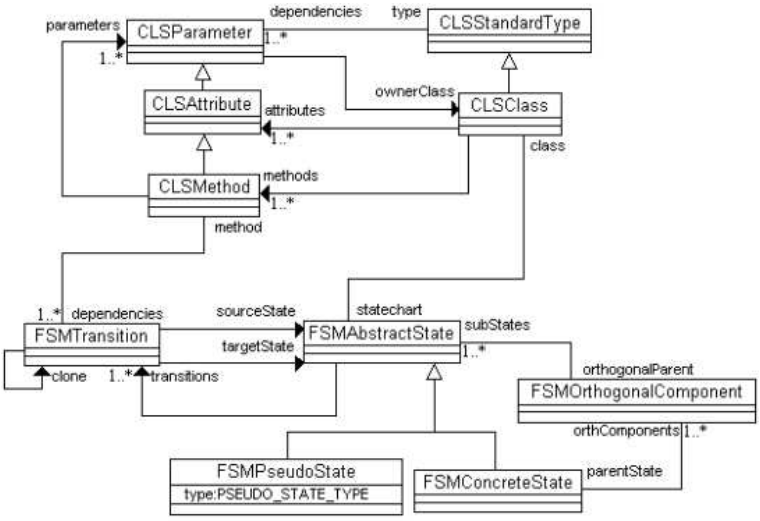


Figure 1. The meta-model of class and state diagrams used in ActiveCASE

Because a state diagram is associated with a class, ActiveCASE supports also a primitive class diagram editor. This kind of diagrams is not an important feature of ActiveCASE, because the tool focuses on behavioural models. However, ActiveCASE can be easily interfaced with other existing CASE tools. The classes that models class diagrams are: **CLSClass** (models a class), **CLSStandardType** (models primitive types like integer, float, string, boolean etc), **CLSParameter**, **CLSAttribute** and **CLSMethod** (model the properties and operations of a specific class).

In ActiveCASE simple states are viewed like composed states with zero substates, and a non-concurrent state like a state, which contains one orthogonal component. This manner of considering state diagrams allows the elimination of redundant classes from meta-model. In the same time, this representation semantically unifies the concepts of simple state, composed state and orthogonal state. Figure 2 shows in a graphical manner the relationships between entities of scalable statecharts.

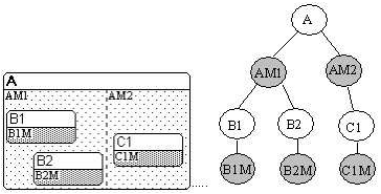


FIGURE 2. Graphical representation of scalable statecharts

**FSMAbstractState** abstract class models all kind of states defined in Level 2 scalable statecharts ($SS^2$). An object of **FSMAbstractState** class contains a list of (incoming and outgoing) transitions (modeled by **FSMTransition** class) and a list of orthogonal components (modeled by **FSMOrthogonalComponent** class). In ActiveCASE there are two state categories: pseudostates (initial, final, history - modeled by **FSMPseudoState** class)and concrete states (modeled by **FSMConcreteState** class). A concrete state contains at least one orthogonal components.

Another important element of our model is that a class has not associated a state diagram or a state machine, but a concrete state. This concrete state represents the parent (root) of all states that describes the behavior of associated class objects. This particular state will have the same name as the modeled class, and its invariant corresponds to the consistency condition imposed on class objects. Practically, the concept of state diagram is not used anymore, and the behavior of objects is described through a state hierarchy.

The auto-transition from **FSMTransition** class level assigns to each transition a 'clone' used in scaling (minimizing or maximizing) composed states. In figure

3 is presented a sample where this double transition is useful. The transitions labeled with m1 and m3 link states from different nesting levels. When *State2* is minimized, its sub states will be 'hidden' and the same thing will happen with their transitions.

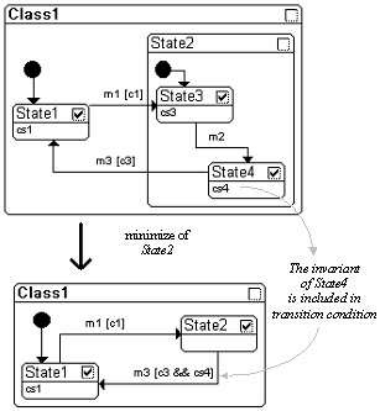


FIGURE 3. Cloning transitions

Therefore, is necessary to introduce supplementary transitions that will preserve the behavioral model. These transitions double the original transitions, and they link states *State1* and *State2*. Also, 'clone' transitions do not influence the source code generation. Their launch conditions are conjunctions between the original condition and the invariant of source state (for the transition labeled with *m3*, its clone will have attached the condition *c3* && *c4*, where && is logical AND operator from C++ programming language).

## 3. ActiveCASE architecture

The ActiveCASE tool has three main components:

- *ActiveCASE.exe* - main application, used for editing class and scalable statecharts diagrams and source code generation,
- *StateControl.ocx* - component used for specific statecharts display,
- *ActiveStatechart.dll* - component used in simulation of active objects behavior during execution of a generated application.

ActiveCASE is a tool for modeling of active objects behavior and offers support for analysis, implementation and testing phases of life cycle of an application. ActiveCASE application allows editing primitive class diagrams and scalable statecharts, and has a C++ source code generator (Figure 4).

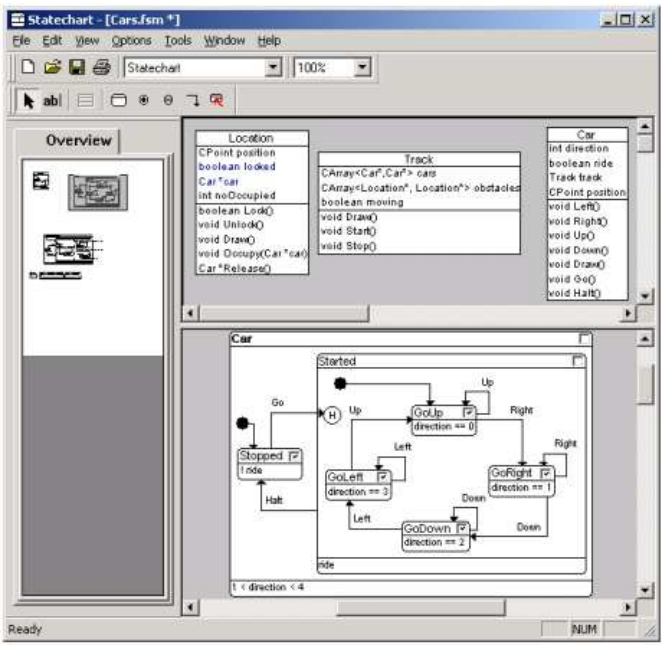


FIGURE 4. ActiveCASE capture with graphical editors of class and states diagrams

All modeled classes are sub-classes of a special class called ActiveObject. This class has attributes and operations for handling states and transitions and for interactions with simulation component.

The component used for simulation allows the visualization of concurrent objects execution during the execution of an application generated by ActiveCASE environment.

## 4. Modeling an application for traffic simulation

In this section is presented a sample application for traffic simulation on a rectangular track. This sample application uses all features provided by ActiveCASE tool and all new elements introduced by scalable statecharts.

Figure 5 shows the class diagram that models an application for traffic simulation on a rectangular track. A **Track** object contains a bi-dimensional array of locations and has a set of associated cars. The three operations of **Track** class allow displaying a **Track** object on screen and to start or stop all its associated cars.

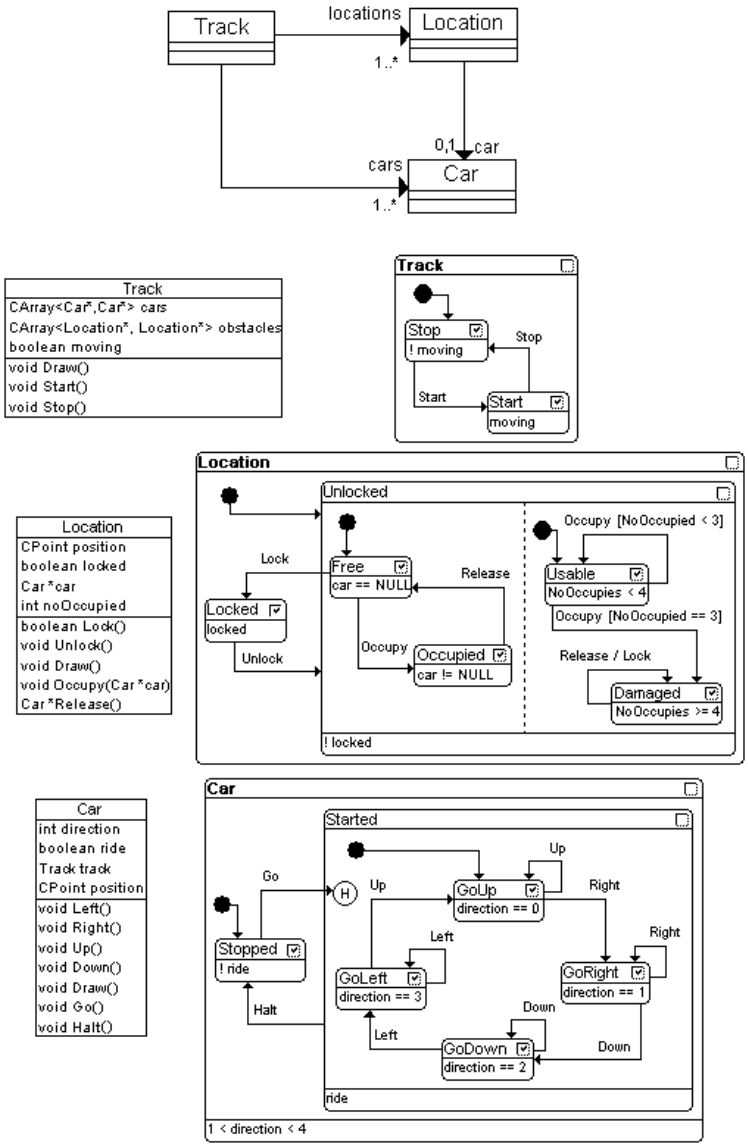


FIGURE 5. State diagrams for **Track**, **Car** and **Location** classes

All diagrams presented in figure 5 are made with ActiveCASE tool. Using ActiveCASE environment the implementation code for all three classes was generated

and attached to a Visual C++ project. The intervention of developer is necessary only for creating a **Track** object and for attaching to it a desired number of cars (**Car** objects). In addition, the developer can implement code for graphical representation of all objects.

Figure 6 shows a test of "Maşină roşie" active object behavior using the simulation of its execution using scalable statecharts.

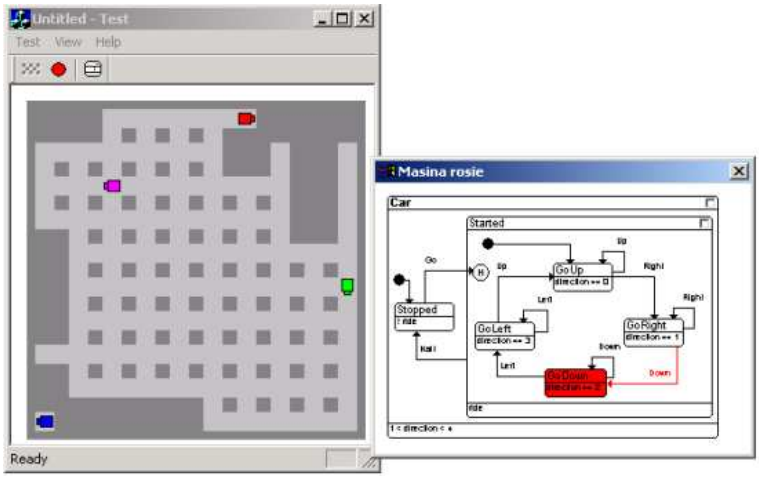


FIGURE 6. The simulation of "Maşină roşie" object behavior

## 5. CONCLUSIONS

The process of concurrent object oriented applications development is laborious. As we stated in the previous sections, the conceptual differences between different concurrent object oriented programming languages make difficult to translate applications from one language in another. In the same time, testing and debugging these applications is more complicated than for sequential applications. Therefore modeling these applications through a unitary set of concepts and notations and testing and debugging them at models level increase the quality of developed applications and decrease the maintenance effort.

The goal of ActiveCASE tool is to automate some steps of the developing process for concurrent object oriented applications. Its main features are:

- flexibility in modeling active objects behavior through scalable statecharts that cover most of concurrent object models;
- high level of internal concurrency specification;
- the source code generator is adaptable to any concurrent object oriented programming language;

- offers support for active objects behavior simulation at run-time;
- imposes an implementation discipline, which ameliorates reuse anomalies.

## References

[1] F. Barbier, H. Briand, B. Dano, S. Rideau, "The Executability of Object-Oriented Finite State Machines", Journal of Object-Oriented Programming, SIGS Publications, 4 (11), pp. 16–24, jul/aug 1998

[2] Michael von der Beeck, "A Comparison of Statecharts Variants", Formal Techniques in Real-Time and Fault-Tolerant Systems, L. de Roever and J. Vytopil (eds.), Lecture Notes in Computer Science, vol. 863, pp. 128–148, Springer-Verlag, New York, 1994

[3] S. Cook, J. Daniels, "Designing Object Systems - Object-Oriented Modelling with Syntropy", Prentice Hall, Englewood Cliffs, NJ, 1994

[4] Bruce Powel Douglas, "UML Statecharts", Embedded Systems Programming, jan. 1999, available at http://www.ilogix.com/fs_prod.htm

[5] D. Harel, A. Naamad, "The STATEMATE Semantics of Statecharts", ACM Transactions on Software Engineering and Methodology, 5 (4), pp. 293–333, 1996

[6] D. Harel, E. Gery, "Executable Object Modeling with Statecharts", IEEE Computer, 30 (7): 31–42, Jul. 1997

[7] David Harel, Statecharts: A Visual Formalism for Complex Systems, Science of Computer Programming, vol.8, no. 3, pp. 231–274, June 1987

[8] Object Management Group, OMG Unified Modeling Language Specification, ver. 1.3, June 1999 available on Internet at http://www.rational.com/

[9] Z. Manna, Mathematical Theory of Computation, McGraw-Hill, 1974

[10] Michael Phillipsen, Imperative Concurrent Object-Oriented Languages, Technical Report TR-95-049, International Computer Science Institute, Berkeley, Aug. 1995

[11] Marian Scuturici, Dan Mircea Suciu, Mihaela Scuturici, Iulian Ober, Specification of active objects behavior using statecharts, Studia Universitatis "Babes Bolyai", Informatica, Vol. XLII, no. 1, pp. 19–30, 1997

[12] Dan Mircea Suciu, Reuse Anomaly in Object-Oriented Concurrent Programming, Studia Universitatis "Babes-Bolyai", Informatica, Vol. XLII, no. 2, pp. 74–89, 1997

[13] Dan Mircea Suciu, Extending Statecharts for Concurrent Objects Modeling, Studia Universitatis "Babes-Bolyai", Informatica, Vol. XLIV, No. 1, pp. 37–44, 1999

[14] Dan Mircea Suciu, Using Scalable Statecharts for Active Objects Internal Concurrency Modeling, Studia Universitatis "Babes-Bolyai", Informatica, Vol. XLV, No. 2, pp. 67–76, 2000

Department of Computer Science, "Babeş-Bolyai" University, 1 M. Kogălniceanu St., RO-3400 Cluj-Napoca, Romania

*E-mail address*: tzutzu@cs.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLVI**, Number 2, 2001

# DESIGNING A FAULT–TOLERANT JINI COMPUTE SERVER

IOAN LAZĂR

Abstract. Java-based tuplespaces provide a simple infrastructure for scientific distributed computing. There are several classes of problems that are not efficiently solvable in JavaSpaces model of computation while efficiently solvable in other tuplespace implementation. JavaSpaces can be used for high performance computing if viewed less strictly in the heritage of Linda and more as a platform-neutral code delivery mechanism.

This paper presents an early design of JavaSpaces compute server. We describe how we use mobile co-ordination and agent wills to provide fault-tolerance in Jini based compute servers. Preliminary experimental results show performance gains made when mobile co-ordination is used.

In this paper we apply the mobile co-ordination and agent wills [12] to provide fault-tolerance in compute servers based on tuple spaces.

The first section the tuple space paradigm and the Linda model for parallel computation. The next section provides an overview of Jini and JavaSpaces [14]. The third section describes in detail our design and then we present our conclusions.

## 1. The Tuple Space Paradigm

Linda is a well known co-ordination model [3]. The fundamental concepts of Linda are tuples, templates and tuple spaces.

A *tuple* is an ordered collection of fields. Each field has a type and a value associated to it. A field with both a value and a type is known as an *actual*. The tuple $< 1989_{int},\ \text{“}Linda\text{”}_{string},\ 1.0_{real} >$ is a tuple containing three fields, with the type of the field shown as a subscript of the value.

Tuples are placed into tuple spaces and are removed from tuple spaces using an associative matching process.

A *template* is similar to a tuple except the fields do not need to have values associated to them, but all fields must have a type. A field that has only a type and no value is known as a *formal*. A template is a tuple which can have formals. A template *matches* a tuple if they have the same number of fields, and

2000 *Mathematics Subject Classification.* 65Y05, 68Q85.

1998 *CR Categories and Descriptors.* F.1.2 [**Computation by Abstract Devices**]: Modes of Computation – *Parallelism and concurrency*; G.4 [**Mathematics of Computing**]: Mathematical Software – D.1.3 [**Programming Techniques**]: Concurrent Programming – E.1 [**Data**]: Data Structures – Distributed data structures.

all the actuals in the template match the actual in the tuple and the formals in the template matches the type of the corresponding actual in the tuple. The templates $< |\Box_{int}, \text{“Linda”}_{string}, 1.0_{real}| >$, and $< |1989_{int}, \text{“Linda”}_{string}, 1.0_{real}| >$ will match the tuple $< 1989_{int}, \text{“Linda”}_{string}, 1.0_{real} >$. ($\Box$ in a template is used to indicate formal fields.)

A *tuple space* is a logical shared associative memory that is used to store tuples. A tuple space implements a *bag* or a *multi-set* (the same tuple may be present more than once and there is no ordering of the tuples in a tuple space).

1.1. **Basic Tuple Space Operations and Matching.** Tuples are inserted into tuple spaces. In order to retrieve a tuple an associative match is performed between a template and the tuples in the tuple spaces. The main primitives (operations) on tuple spaces are:

***out(tuple)*:** Place the *tuple* into the tuple space.

***in(template): tuple*:** Removes a *tuple* from the tuple space. The *tuple* removed is associatively matched using the template and the *tuple* is returned to the calling process. If not tuple that matches exists then the calling process is blocked until one becomes available.

***rd(template): tuple*:** This primitive is identical to *in* except that the matched tuple is not removed from the tuple space, and a copy is returned to the calling process.

***eval(active tuple)*:** The *active tuple* contains one or more functions, evaluated in parallel with each other and the calling process. When all the functions have terminated, a tuple is placed into the tuple space with the results of the functions as its elements.

1.2. **Multiple rd problem.** The *rd* operation returns one arbitrary matching tuple from all tuples matching a given template. If a process wants to iterate over the list of all tuples matching a given template, no atomic Linda operation is provided in order to do this.

One solution presented in [10] is adding another operation *copy-collect* to the Linda operations. This operation reads all matching tuples in the tuple space and copies them into another tuple space. Therefore this extension of the tuple space model needs multiple addressable tuple spaces [3].

A similar operation *collect* is added to solve the equivalent multiple *in* problem, which moves all matching tuples from one tuple space to another tuple space.

Another solution is to change the application using the tuple space instead. One way is to include an index field in the tuples and iterate over that field when *rd* operations.

1.3. **Predicate Operations.** Linda includes predicate variants of the two operations *in* and *rd*, named *inp* and *rdp*. These are non-blocking versions of the

operations, meaning that if no tuple matches the template provided these operations return the boolean value *false* (or some other value) indicating that no matching tuple was found, and do not block.

In general, the non-blocking Linda operations *rdp* and *inp* cause a problem because they inspect the "present" state of a tuple space, and one could argue that they are inappropriate in a distributed environment, where "the most recent operation" is not defined [7].

1.4. **Fault-tolerant tuple spaces.** Early tuple space based languages suffered from poor fault agent (program, process) fault tolerance. Later, many systems have used transactions to provide fault-tolerant Linda primitives: PLinda [6], JavaSpaces [14] and TSpaces [16]. More recently Rowstron [12] proposed a new technique *mobile coordination* in order to provide fault tolerant tuple-space based co-ordination.

**Transactions.** Most implementations which use transactions add two new primitives which are *start* and *commit* [12]. The *start* primitive causes the server managing the tuple spaces to retain all copies being removed and to hold all copies being inserted by the agent (program, process) which performed the *start*. When the agent execute the *commit* operation, the tuples inserted under the transaction are actually placed in the tuple space and the tuples deleted are actually discarded. There are many problems with this approach [12]: altering the semantics of the co-ordination language and how can decide the server if the agent that performed a *start* operation is alive?

As an alternative to this traditional approach using transactions the mobile co-ordination approach solves the problems mentioned above.

**Mobile Co-ordination.** In this approach the co-ordination primitives are moved on the server which store the tuple space. If all the coordination *primitives* reach the server before any are executed then the entire operation of the agent will be executed.

An overview of this framework from an application developer perspective is as follows (see Figure 1). The agent who whish to execute an operation composed by many primitives *in* and *out*, encapsulate these primitives into the function *coordination*. This function will be executed by the tuple-space server. In order to migrate this code to the tuple-server, the application developer creates a class that implements the *MobileCoordination* interface (extended from the interface *Serializable*). After that, the agent calls the tuple-space operation *executeSafe* which returns a tuple.

An *agent will* is a set of tuple space access primitives that are executed when the tuple space server managing the tuple spaces decides that an agent owning the will has failed.

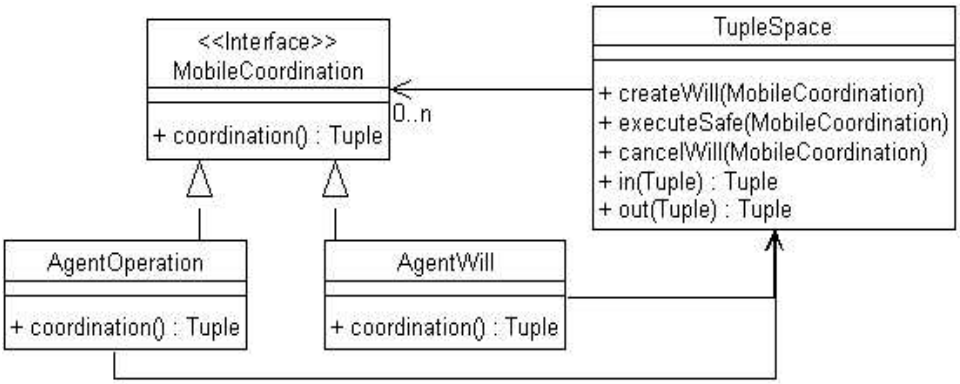


FIGURE 1. Mobile coordination class diagram

The following lines of code shows how an agent can take safely a tuple from a tuple space:

```
AgentOperation inOperation = new AgentOperation();
AgentWill inWill = new AgentWill(inOperation);
tuplespace.createWill(inWill);
tuple = tuplespace.executeSafe(opIn);
tuplespace.cancelWill(inWill);
```

where

```
class AgentOperation implements MobileCoordination {
  Tuple t;
  Tuple coordination() { //operations executed on server
    t = tuplespace.in(template);
    return t;
  }
}
class AgentWill implements MobileCoordination {
  Tuple t = null;
  AgentWill(AgentOperation inOperation) {
    this.tuple = inOperation.tuple;
  }
  Tuple coordination() { //operations executed on server
    tuplespace.out(tuple);
    return null;
  }
}
```

```
class TupleSpace {
  Tuple executeSafe(MobileCoordination agent) {
    return agent.coordination();
  }
}
```

This fault-tolerant mechanism is working if the `AgentOperation` and `AgentWill` objects are migrated to the tuple space server and their `coordination` methods are executed there.

1.5. **Tuple Space Usage in Parallel Computing.** The tuple space paradigm provides a mechanism for communication between processes in a distributed system. The abilities provided by a tuple space are for processes to *share data* and *coordinate events* in a distributed environment.

The data sharing of a tuple space is simply achieved by having several processes accesing a tuple space as a global shared memory.

The event coordination ability of a tuple space is achieved using a tuple to simulate a semaphore by having clients insert tuples into a tuple space and withdraw tuples from a tuple space. A single-element tuple is functionally equivalent to a semaphore [4]. The coordination feature of the tuple space is illustrated by implementing a counting $n$ semaphore:

*begin*
    $in(t_{\text{semaphore}})$
        //critical region
    $out(t_{\text{semaphore}})$
*end*

where the initial value $n$ can be obtained by $n$ repetitions of $out(t_{semaphore})$.

There are two types of coordination (or synchronization): mutual exclusion and conditional synchronization. Mutual exclusion is exemplified in the pseudo-program above. Conditional synchronization – waiting for an event to occur, is also an integral part of Linda. This is achieved by blocking operations – waiting until a tuple is present.

## 2. Design overview of Jini and JavaSpaces

The Jini technology [15] is a runtime infrastructure that resides on the network and provides mechanisms to enable addition, removal, discovery and access of services. Jini enable building and deploying distributed systems that are organized as a *federation of services*. A *service* is an entity capable of performing some function. Services advertise their capabilities via a look up server. The primary function of lookup servers is to assist Jini enabled clients to discover and access services.

| | |
|---|---|
| *out(...)* | `write(Entry entry, ...)` |
| *in(...)* | `take(Entry template, ...)` |
| *rd(...)* | `read(Entry template, ...)` |
| *eval(...)* | not included |
| *inp(...)* | `takeIfExists(Entry template, ...)` |
| *rdp(...)* | `readIfExists(Entry template, ...)` |
| not included | `write(Entry, ...)` |

TABLE 1. Linda and JavaSpaces operations and terminology

JavaSpaces is a Java implementation of a tuple-based system [14], and is provided as a service based on Jini technology.

The tuple space is represented by the `JavaSpace` interface, and the Linda tuples and templates are represented by the `Entry` marker interface.

The `JavaSpace` operations `read`, `take` and `write` correspond to the *rd*, *in* and *out* Linda primitives. All these operations are synchronous (blocking operations).

There are also two asynchronous (non-blocking) operations `readIfExists` and `takeIfExist`. These types of asynchronous operations correspond to some Linda primitive extensions [11].

In order to provide fault tolerance for tuple space primitives, the `read`, `take` and `write` operations can be performed as part of a transaction or not. The description of the basic primitives include descriptions of how they interact with transactions. This fact increases the complexity of the implementation, and makes it from a simple model into a complex one.

JavaSpaces does not introduce any new concept at the Linda model level. JavaSpaces objects (`Entry` derived objects) can be introduced in a space but there they are not active objects. The `JavaSpace` interface does not have any operation corresponding to Linda *eval* primitive.

## 3. A Framework for Adaptive Master-Worker Parallelism

Master-worker parallelism is a widely used form of parallel application programming [13, 2]. It is conceptually very simple and involves dividing a problem into a smaller number of independent work units which can be distributed to remote worker processes for computation in parallel. A single master process centrally controls both the distribution of work units to worker processes and the returned of computed results back to the master process. The method of maintaining a collection of work units is referred as work queue or task farm scheduling.

There are many opportunities for running distributed running applications [13]. We choose here the Jini [15] environment for the master and worker processes, and JavaSpaces for task scheduling.

A typical space-based compute server works as described below:

- A task is an entry that both describes the specific of the task and contains methods that performs the necessary computations.
- Worker processes monitor a space, take tasks as they become available, compute them, and then write their results back to the space.
- Results are entries that contain data from computation's output.

Spaced-based computer servers have the following nice properties:

**Scalability:** the more worker processes there are, the faster the tasks will be computed. Workers can be added or removed at run time and the computation will continue as long as there is at least one worker to compute tasks.

**Load balancing:** workers running on slower CPU will compute their tasks slowly (and thus complete fewer tasks) than those running on faster CPU.

**Low coupling:** the master and the workers are uncoupled. The workers do not have to know anything about the master and the specific of the task - they just compute them and return results to the space.

### 3.1. **Basic abstractions.**

**Tasks.** Our spaces hold tasks. The task is considered an *active entity* of the space if it is not completed.

The `Task` class implements the Jini `Entry` interface in order to be JavaSpaces compatible. Also, the `Task` class defines a method `compute()` that is overridden by user-defined tasks and a method `execute()` called by the `Worker` processes. The `compute()` method should be an abstract method but in order the `Task` to be a template for retrieving tuples from the JavaSpaces spaces, the `Task` must be a concrete class.

The master process writes `Task` instances (`done` = `False`) into spaces and the workers take tasks `execute` them, and then return the completed tasks back into the space (`done` = `True`).

**Master and Worker Processes.** Master and worker processes use Jini services for JavaSpaces and distributed transactions. In the current implementation of the framework both uses a `TransactionManager` and a `JavaSpace` Jini services. This services are available through some methods of `AbstractProcess` class.

The abstract methods `generateTasks()` and `collectResults()` of the class `GenericMaster` are defined in concrete master processes.

A worker continually looks for tasks, takes it from a space, computes it and writes the result back in a space. The significant running code for the worker process is:

```
work() {
  for ( ; ; ) {  //looks continually for tasks
    Task task = taskReader.takeTask();  //take (safe) a task
```

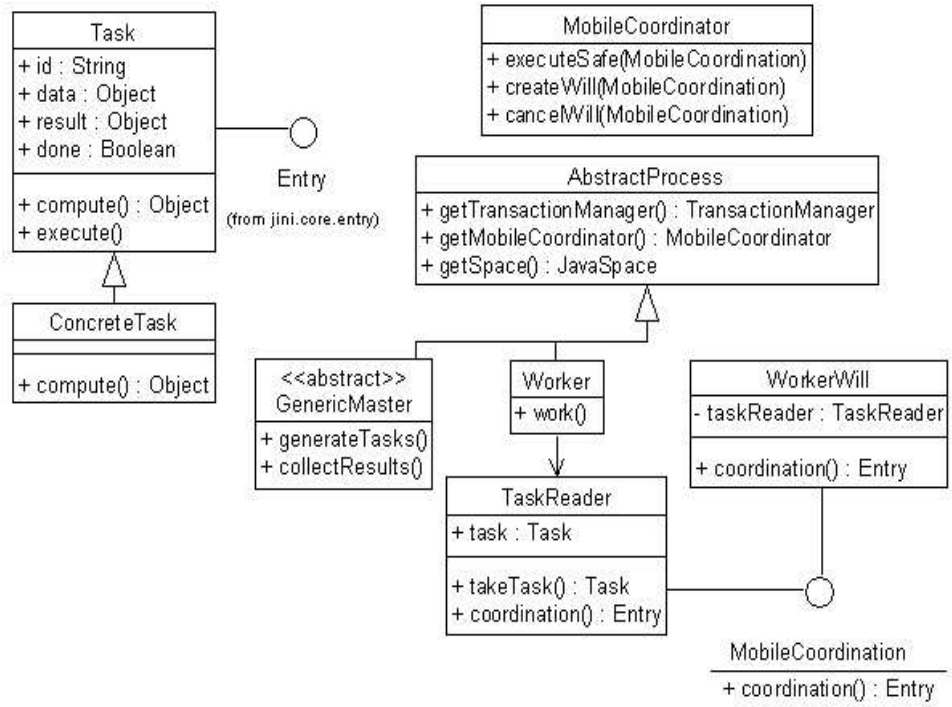


FIGURE 2. A Master-Worker Jini Framework

```
      task.execute();                    //execute, and then
      getSpace().write(task);       //write the task
    }                               //(completed) back to the space
  }
```

where `taskReader` is an instance of the class `TaskReader`.

The next subsection discusses the rest of the design (our fault–tolerant decisions).

3.2. **Fault Tolerance.** The worker process uses mobile co-ordination and agent wills in order to provide a fault tolerant solution in our compute framework.

The `TaskReader` class encapsulates a safe Linda *in* operation on a JavaSpaces space. The `takeTask` method returns a `Task` from the space and it is a blocking operation:

```
1.  Task takeTask() {
2.    WorkerWill will = new WorkerWill(this);
```

```
3.    mobileCoordinator.createWill(will);
4.    Task task = mobileCoordinator.executeSafe(this);
5.    mobileCoordinator.cancelWill(will);
6.  }
```

The `mobileCoordinator` in the above code is a proxy of a Jini service running in the same Java virtual machine as the JavaSpaces service. (We have made some minor modifications in the `com.sun.jini.outrigger.SpaceProxy` class, but not altering the JavaSpaces services.) Line 4 causes the `TaskReader` object to be migrated to the server, and the result is returned.

The `MobileCoordinator` server executes the following:

```
Entry executeSafe(MobileCoordination mc) {
  return mc.coordination();
}
```

which is the `TaskReader`'s code:

```
Entry coordination() {
  this.task = space.take(new Task(), null, Long.MAX_VALUE)
  return task;
}
```

but executed into the server.

Now, if the worker dies while executing the task, then the server can restore the task back into the space since the `Worker` *will*, encapsulated into the class `WrokerWill`, is:

```
Entry coordination() {
  if (taskReader.task != null)
   space.write(taskReader.task, null, Long.MAX_VALUE);
  return null;
}
```

where the `taskReader` is a copy (stored in the server) of the `TaskReader` object.

## 4. Conclusions

In this paper we have demonstrated how the concepts of mobile co-ordination can be used to provide fault–tolerant Jini compute servers.

This paper describes the use of a single tuple space server. In real implementations, multiple servers must be used. A comparison between the transactions model used in Jini and also a multiple fault–tolerant tuple space servers is an area under consideration.

We also will investigate the proposed design related to some computational problems – linear and nonlinear solvers.

## References

[1] N. Carriero, E. Freeman, G. Gelernter, D. Kaminsky, *Adaptive parallelism and Piranha*, IEEE Computer, 28(1):40-49, 1995.

[2] N. Carriero, G. Gelernter, *How to write parallel programs: a first course*, MIT Press, Cambridge, 1990.

[3] N. Carriero, D. Gelernter, Linda in context, *Communication of the ACM*, 32(4):444–458, 1989.

[4] D. Gelernter, Generative Communication in Linda, *ACM Transactions on Programming Languages and Systems*, 7(1), 1985.

[5] E. Freeman, S. Hupfer, K. Arnold, *JavaSpaces Principles, Patterns and Practice*, Addison Wesley, 1999.

[6] K. Jeong, D. Shasha, Persistent Linda 2: a transaction/checkpointing approach to fault-tolerant Linda, in *Proc* $13^{th}$ *Symposium on Fault-Tolerant Distributed Systems*, 1994.

[7] J.E. Larsen, J.H. Spring, *GLOBE: Global Object Exchange*, Candidatus Scientiarum in Computer Science Thesis, Univ. Copenhagen, 1999.

[8] M.S. Noble, S. Zlateva, *Distributed Scientific Computation with JavaSpaces?*, Boston University, Technical Report CN01-34, 2001.

[9] A. Rowstron, Using Agent Wills to Provide Fault-tollerance in Distributed Shared Memory Systems, $8^{th}$ *EUROMICRO Workshop on Parallel and Distributed Processing*, Rhodos, Greece, IEEE Press, pp. 317-324, 2000.

[10] A. Rowstron, A.M. Wood, Solving the Linda multiple rd problem, *Coordination Languages and Models, Proc. Coordination '96*, eds. P. Ciancarini, C. Hankin, Springer-Verlag, LNCS 1061, 1996, pp. 357–367.

[11] A. Rowstron, Using asyncronous tuple space access primitives (BONITA primitives) for process co-ordination, *Coordination Languages and Models*, eds. D. Garlan, D. Le Metayer, Springer-Verlag LNCS 1282, pp. 426–429, 1997.

[12] A. Rowstron, Mobile Co-ordination: Providing fault tolerance in tuple space based co- ordination languages, *Coordination Languages and Models, Coordination '99*, eds. P. Ciancarini, P. Wolf, Springer-Verlag, LNCS 1594, 1999, pp. 196–210.

[13] G. Shao, *Adaptive Scheduling of Master/Worker Applications on Distributed Computational Resources*, Phd Thesis, Univ. California, San Diego, 2001.

[14] Sun Microsystems, *Jini Specifications*, Available from Sun Microsystems WWW Site (http://java.sun.com/products/javaspaces/), 1998.

[15] Sun Microsystems, Jini Specifications, Available from Sun Microsystems WWW Site (http://www.sun.com/jini/specs/), 2000.

[16] P. Wyckoff, S. McLaughry, T. Lehman, D. Ford, TSpaces, *IBM System Journal*, 1998. 1994.

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

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

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLVI**, Number 2, 2001

# NUMERICAL SOLUTION OF THE DELAY DIFFERENTIAL EQUATIONS BY NONPOLYNOMIAL SPLINE FUNCTIONS

V. A. CĂUŞ AND G. MICULA

Abstract. In this paper the nonpolynomial spline function to approximate the solution of the delay differential equations is constructed. The stability and the convergence of the nonpolynomial spline algorithms are also investigated.

Key Words: nonpolynomial splines, delay equations, collocation.

## 1. Introduction

The delay differential equations provide realistic models for many phenomena arising in applied mathematics. As known, delay differential equations can be used for the modeling of population dynamics, the spread of infections diseases, two-body problems of electrodynamics, etc.

Delay differential equations, or generally functional differential equations have been extensively studied in the past decades, especially as models to describe many physical and biological systems. Although, there exist several methods to solve numerically the delay differential equations, most of them cannot handle some difficulties properly. At the same time spline functions have steadily advanced to the front position as a very useful tool in general for the approximate the solutions of nonlinear differential, integral and partial differential equations, and particularly of the modified argument differential equations.

Comprehensive bibliography of published papers in this field (see [15]) and especially the reference therein.

The aim of this paper is to propose an alternative approximate method for the numerical solution of the delay differential equation problem and to introduce a new approach to the stability analysis of the nonpolynomial spline approximate

2000 *Mathematics Subject Classification.* 65L05,65D05,65L70,34K10.

1998 *CR Categories and Descriptors.* G.1.7 [**Mathematics of Computing**]: Numerical Analysis – *Ordinary Differential Equation*; G.1.1 [**Mathematics of Computing**]: Numerical Analysis – *Interpolation.*

solution. We will only consider in this paper the following scalar delay differential equation problem:

$$\begin{aligned} y'(t) &= f(t, y(t), y(g(t))), t \in [0, T] \\ y(t) &= \varphi(t), y'(t) = \varphi'(t), t \in [\alpha, 0], \alpha < 0 \end{aligned} \tag{1}$$

Several other methods for such delay differential equation problems have been proposed and their convergence has been investigated. For example, Oberle and Pesch [16] investigated the convergence of a numerical method for a constant delay $g(t) = t - h$, Bellen and Zennaro [4] investigated the convergence of a numerical method for time dependent delay, Tavernini [18], Arndt [2], Feldstein and Neves [8], Karoui and Vaillancourt [12, 13] and Baker and Paul [3] investigated the convergence of a numerical methods for state dependent delay differential equations. Jackiewicz [9, 10, 11] investigated the convergence of numerical method for time dependent delay of neutral delay differential equations. Recently, Enright and Hayashi [7] gave a very deep investigation of the convergence analysis of the solution of retarded and neutral delay differential equations by continuous numerical method.

We assume the existence, uniqueness and stability of the solutions to the mathematical problem under consideration. For example, sufficient conditions for the existence and uniqness of solution to the delay differential equation problem (1) are:

-$f$ is continuous with respect to $t, y(t), y(g(t))$,

-$y(t)$ is continuous,

-$f$ satisfies a Lipschitz condition in the last two argument,

-$\varphi$ is continuous and

-$f$ is bounded (see Driver[6]).

We suppose here that $f \in C^r([0, T] \times R^2, R)$ for a given natural number $r \in N$ and we shall introduce the nonpolynomial spline function space of degree $m \in N$ denoted by $S_m(\Delta)$, in which we shall find an approximate solution for the problem (1). It will be shown that our approximating method is a one-step method and the order of the method is $O(h^{\beta+r+m})$ in $y^{(q)}, (q = 0, 1, ..., r + 1), 0 < \beta \leq 1$.

Here $m$ is an arbitrary positive integer, which in fact indicates the number of iteration processes in the method that describes the spline function.

Assume that $f$ satisfies the following Lipschitz condition:

$$\left| f^{(q)}(t, u_1, v_1) - f^{(q)}(t, u_2, v_2) \right| \leq L \left[ |u_1 - u_2| + |v_1 - v_2| \right] \tag{2}$$

where $(t, u_1, v_1), (t, u_2, v_2) \in [0, T] \times R^2$.

The continuity of $f$ and the Lipschitz condition (2) guarantee the existence and uniqueness of the solution $y : [\alpha, T] \to R$ of problem (1).

Assume that the delay function g satisfied the condition $g(t) \leq t, t \in [\alpha, 0]$ and the jump discontinuities to be known for sufficiently high-order derivatives of $y$ and are given in the form:

$$\Delta : \xi_0 < \xi_1 < ... < \xi_M \tag{3}$$

We will construct a nonpolynomial spline function $s : [0, T] \to R$ in such a way that on each interval $[\xi_k, \xi_{k+1}]$ $s$ be a nonpolynomial spline function.

We will use a collocation method of the order $O(h^{\beta+r+m})$ in $f^{(q)}, q = 0, 1, ..., r+1$. The function $f^{(q)}, q = 0, 1, ...$ is a function of the variables $t, y(t)$ and $y(g(t))$ and it will be obtained from the following algorithm:

If we denote $f^{(0)} := f\left(t, y(t), y(g(t))\right)$, then for all $q = 0, 1, 2, ...$

$$y^{(q+1)} := \frac{d^q f}{dt^q} := f^{(q)} := \frac{\partial^{(q-1)} f}{\partial t} + \frac{\partial^{(q-1)} f}{\partial y(t)} \cdot f + \frac{\partial^{(q-1)} f}{\partial y(g(t))} \cdot \frac{\partial y(g(t))}{\partial g(t)} \cdot \frac{\partial g(t)}{dt} \tag{4}$$

can be used as a recurrence formula. Let us consider the first interval $[\xi_0, \xi_1]$ which is [0,$\xi_1$] and the uniform partition of this interval

$$\xi_0 = t_0 \leq t_1 \leq ... \leq t_{m-j+1} \leq ... \leq t_m \leq \xi_1 \tag{5}$$

Choosing a sufficiently large arbitrary positive integer $m$, let us define the nonpolynomial spline functions $s$, which approximate the solution $y$ of (1)

$$s(t) := s_k^{[m]}(t) + \int_{t_k}^{t} f[t_1, s_k^{[m-1]}(t_1), s_k^{[m-1]}(g(t_1))]dt_1 \tag{6}$$

on the subinterval $t_k \leq t \leq t_{k+1}, (k = 1, 2, ..., n-1)$ such that
$s_{-1}^{[m]}(t_0) = \varphi(t_0), s_0(g(t)) = \varphi(g(t))$, and $s_0(t) = \varphi(t), t \in [\alpha, 0]$

Define the nonpolynomial spline function $s_0$ approximating the solution $y$ of (1), on the first interval $I_1 : [t_0, \xi_1]$ by

$$s_0(t) = \varphi(t_1) + \int_{t_0}^{t} f(t_1, \varphi^{[j-1]}(t_1), \varphi^{[j-1]}(g(t_1)))dt_1 \tag{7}$$

Associating the following $m$ iteration processes

$$s_1(t) = s_0^{[m]}(t_1) + \sum_{j=0}^{r} \frac{(t-t_1)^{j+1}}{(j+1)!} f^{(j)}(t_1, s_0^{[m]}(t_1), s_0^{[m]}(g(t_1))) \tag{8}$$

Let us denote the nonpolynomial spline function by $s_k, s_k \in S_m$, which is approximating the solution on the interval $I_k : [\xi_{k-1}, \xi_k]$. $s_k$ is a nonpolynomial spline function such that $s_k : [\xi_{k-1}, \xi_k] \to R$. In a similar manner, following the introduced procedure one can easily construct spline functions on each subinterval $I_k : [\xi_{k-1}, \xi_k]$.

$$\begin{aligned} s_k(t) &= s_{k-1}(t_k) + \\ &+ \int_{t_{k-1}}^{t} f(t_{m-j+1}, s^{[j-1]}(t_{m-j+1}), s^{[j-1]}(g(t_{m-j+1})))dt_{m-j+1} \end{aligned} \tag{9}$$

and

$$s_k(t) = s_{k-1}^{[m]}(t) + \sum_{j=0}^{r} \frac{(t-t_k)^{j+1}}{(j+1)!} f^{(j)}(t_k, s_{k-1}^{[m]}(t_k), s_{k-1}^{[m]}(g(t_k))), \tag{10}$$

for $(k = 0, 1, ..., n-1)$.

We call the space $S_m(\Delta) = \{s$ :there exists polynomials, $s_0, s_1, ..., s_n$ such that $s(x) = s_i(x)$ for $x \in I_i, (i = 1, 2, ...)$ and $D^j s_{i-1}(x_i) = D^j s_i(x_i)$ for $j = 0, 1, 2, ....\}$

Here the derivatives $s^{(j)}$ are left-hand limits of the segment of $s$ defined on $[t_{k-1}, t_k]$.

This procedure yields a spline function $s \in S_m$ over the entire interval $[\xi_j, \xi_{j+1}]$ with the knots $\{t_k\}_{k=1}^{N}$.

By construction its obvious that $s \in C^r([\xi_j, \xi_{j+1}], R)$. Thus the exact solution of problem (1) can be written in the following form on the interval $I_1$ :

$$y(t) := y_k^{[m]}(t_k) + \int_{t_0}^{t} f(t_1, y^{[m-1]}(t_1), s^{[m-1]}(g(t_1)))dt_1 \tag{11}$$

where the following $m$-iterations processes are considered

$$y(t) := y^{[m]}(t_k) + \sum_{j=1}^{r} \frac{(t-t_k)^j}{j!} y^{(j)}(t_k) + \frac{y^{(r+1)}(\eta_k)}{(r+1)!} (t-t_k)^{r+1} \tag{12}$$

$$\begin{aligned} y(t) &: = y^{[j]}(t_k) + \\ &+ \int_{t_k}^{t} f(t_{m-j+1}, y^{[j-1]}(t_{m-j+1}), s^{[j-1]}(g(t_{m-j+1})))dt_{m-j+1} \end{aligned} \tag{13}$$

such that $j = 1, 2, ...m, t_k \leq \eta_k \leq t_{k+1}$ and $k = 0, 1, 2, ...$

It is then clear that the continuity of $f$ and the Lipschitz condition (2) guarantee the existence and uniqueness of the solution (1) on every subinterval $[t_k, t_{k+1}]$.

## 2. Error estimation and Convergence

We show here that the global error of introduced numerical solution method for delay differential equations is bounded on the whole interval.

**Theorem 1.** *Assume that $f$ satisfies the Lipschitz condition (2) and $s(t)$ is the nonpolynomial spline approximation of the solution of (1). Then the order of the introduced method is $O(h^{\beta+r+m})$.*

**Proof.** Let denote $L = \max\{L_1, L_2, ..., L_m\}$ as a Lipschitz constant and consider the first interval $I_1 : [\xi_0, \xi_1]$. For the error estimation of the introduced method using Lipschitz condition we get

$$\begin{aligned} |y(t) - s(t)| &\leq L \int_{t_0}^{t} \{ \left| y^{[m-1]}(t_1) - s^{[m-1]}(t_1) \right| + \\ &\quad + \left| y^{[m-1]}(g(t_1)) - s^{[m-1]}(g(t_1)) \right| \} dt_1 \end{aligned}$$

$$\begin{aligned} &\leq 2L^2 \int_{t_0}^{t} \int_{t_0}^{t_1} \{ \left| y^{[m-2]}(t_2) - s^{[m-2]}(t_2) \right| + \\ &\quad + \left| y^{[m-2]}(g(t_2)) - s^{[m-2]}(g(t_2)) \right| \} dt_2 dt_1 \end{aligned}$$

$$...$$

$$\begin{aligned} &\leq 2^{m-1} L^m \int_{t_0}^{t} \int_{t_0}^{t_1} ... \int_{t_0}^{t_{m-1}} \{ \left| y^{[0]}(t_m) - s^{[0]}(t_m) \right| + \\ &\quad + \left| y^{[0]}(g(t_m)) - s^{[0]}(g(t_m)) \right| \} dt_m ... dt_1 \end{aligned} \tag{14}$$

and consequently we obtain

$$|y(t)-s(t)| \leq \frac{2^m L^m}{(r+m+1)!} h^{r+m+1} \cdot W(h) = O(h^{r+m+1}) \tag{15}$$

where $W(h) = \max\left\{\omega(y^{(r+1)}(t), h)\right\}$ such that $\omega(y^{(r+1)}, h)$ is the continuity moduli of the function $y^{(r+1)}$ and $h$ is the step-length..

Similarly the difference $|\acute{y}(t) - \acute{s}(t)|$ can be estimated easily. Thus

$$|\acute{y}(t)-\acute{s}(t)| \leq L[\left|y^{[m-1]}(t)-s^{[m-1]}(t)\right| + \left|y^{[m-1]}(g(t))-s^{[m-1]}(g(t))\right|]$$

$$\leq 2L^2 \int_{t_0}^{t} [\left|y^{[m-2]}(t_2)-s^{[m-2]}(t_2)\right| + \left|y^{[m-2]}(g(t_2))-s^{[m-2]}(g(t_2))\right|]dt_2$$

$$\leq 2^{m-1}L^m \int_{t_0}^{t}\int_{t_0}^{t_2} ... \int_{t_0}^{t_{m-1}} [\left|y^{[0]}(t_m)-s^{[0]}(t_m)\right| +$$

$$+ \left|y^{[0]}(g(t_m))-s^{[0]}(g(t_m))\right|]dt_m dt_{m-1}...dt_2$$

and finally we obtain

$$|\acute{y}(t)-\acute{s}(t)| \leq \frac{2^m L^m}{(r+m)!} h^{r+m} \cdot W(h) = O(h^{\beta+r+m}) \tag{16}$$

Hence choosing for $q = 2, 3, ..., r+1$ we get

$$\left|y^{(q)}(t)-s^{(q)}(t)\right| \leq \frac{2^k L^k}{(r+m)!} h^{r+m} \cdot W(h) = O(h^{\beta+r+m}) \tag{17}$$

This completes the proof of the theorem. Let denote the difference

$$e(t) = |y(t)-s(t)|, e'(t) = |\acute{y}(t)-\acute{s}(t)|. \tag{18}$$

**Lemma 2.** *Suppose $f \in C^r([0,T]\times[\alpha,T]\times[\alpha,T],R), r \in N$ and $f$ satisfy the Lipschitz condition (2) with a constant $L = \max\{L_1, L_2, ..., L_m\}$, then there exist constants $C_1$ and $C_2$ which are independent of $h$ such that*

$$e(t) \leq C_1 h^{r+m} W(h) = O(h^{\beta+r+m})$$

$$e'(t) \leq C_2 h^{r+m} W(h) = O(h^{\beta+r+m}) \tag{19}$$

where $0 < \beta \leq 1$. Similarly it can be easily shown that there exists a constant $C_3$ which is independent of $h$ such that the following inequality holds

$$\left|y^{(q)}(t) - s^{(q)}(t)\right| \leq C_3 h^{r+m} W(h) = O(h^{\beta+r+m}) \tag{20}$$

where $q = 2, 3, ..., r+1$ and $t \in [t_k, t_{k+1}]$. Proof of Lemma is obvious.

From (20) and defined procedure we have the following subsequent assertion:

**Theorem 3.** *Let $y : [0,T] \to R$ be the exact solution of the problem (1). If $s : [0,T] \to R$ is the nonpolynomial spline approximation of the solution of (1), defined by the introduced procedure, then the following inequalities hold:*

$$|y(t) - s(t)| \leq \frac{C_0}{(r+m+1)!} h^{r+m+1} W(h) \tag{21}$$

$$\left|y^{(q)}(t) - s^{(q)}(t)\right| \leq \frac{C_0}{(r+m)!} h^{r+m} W(h) \tag{22}$$

where $C_0 = 2^m L^m, q = 0, 1, 2, ..., r+1$, for all $t \in [t_k, t_{k+1}]$ and consequently

$$\left|y^{(q)}(t) - s^{(q)}(t)\right| \leq C h^{r+m} W(h) \tag{23}$$

holds for all $t \in [t_k, t_{k+1}]$ and for $q = 0, 1, 2, ..., r+1$. Here $C$ is a constant independent of $h$.

## References


[1] AHLBERG J.H., NILSON E.N. and WALSH J.L. - The theory of Splines and Their Applications, Academic Press, New York, 1967

[2] ARNDT H. - Numerical solution of retarded initial value problems: Local and global error and stepsize control, Numer.Math., 43(1984), pp.343-360

[3] BAKER C.T.H. and PAUL C.A.H. - Parallel continuous Runge-Kutta Methods and vanishing lag delay differential equations, Adv. Comput. Math., 1(1993), pp.367-394

[4] BELLEN A. and ZENNARO M. - Numerical solution of differential equations by uniform corrections to an implicit Runge-Kutta method, Numer.Math., 47(1985), pp.301-316

[5] BLAGA P., MICULA G. and AKCA H.- On the use of spline function of even degree for the numerical solution of the delay differential equations, CALCOLO, 1-2/32(1990)

[6] DRIVER R.D. - Ordinary and Delay Differential Equations, Springer Verlag, Berlin, 1977

[7] ENRIGHT W.E. and HAYASHI H. - Convergence analysis of the solutions of retarded and neutral delay differential equations by continuous numerical methods, SIAM J. Numer. Anal. 35(1998)

[8] FELDSTEIN A. and NEVES K.W. - High order methods for state-dependent delay differential equations with nonsmooth solutions, SIAM J. Numer. Anal. 21(1984)

[9] JACKIEWICZ Z. - One-step methods of any order for neutral functional differential equations, SIAM J. Numer. Anal. 23(1986)
[10] JACKIEWICZ Z. - Variable-step variable-order algorithm for the numerical solution of neutral functional differential equations, Appl. Numer.Math, 3(1987)
[11] JACKIEWICZ Z. - The numerical solution of neutral functional differential equations by Adams predictor-corector methods, Appl. Numer.Math, 8(1991)
[12] KAROUI A. and VAILLANCOURT R.-Computer solutions of state-dependent delay differential equations, Comput.Math.Appl, 27(1994)
[13] KAROUI A. and VAILLANCOURT R. - A numerical method for vanishing-lag delay differential equations, Appl. Numer.Math, 17(1995)
[14] MICULA G. and AKCA H. - Numerical solutions of differential equations with deviating argument using spline functions, Studia Univ. Babes-Bolyai, Mathematica 33(1988)
[15] MICULA G. and MICULA S. - Handbook of splines, Kluwer Academic Publishers, Dordrecht-London-Boston, 1999
[16] OBERLE H.J. and PESCH H.J. - Numerical treatment of delay differential equations by Hermite interpolation, Numer.Math., 37(1981)
[17] SCHUMAKER L.L. - Spline Functions: BasicTheory, John Wiley and Sons inc., New-York-Chichester-Brisbane-Toronto, 1981
[18] TAVERNINI L. - The approximate solution of Volterra differential systems with state-dependent time legs, SIAM I.Numer.Anal. 15(1978)

University of Oradea, Romania
*E-mail address*: vcaus@uoradea.ro

Babeş-Bolyai University of Cluj-Napoca, Romania
*E-mail address*: ghmicula@math.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLVI**, Number 2, 2001

# A NEW ALGORITHM FOR WORD SENSE DISAMBIGUATION

DOINA TĂTAR AND GABRIELA ŞERBAN

Abstract. The task of disambiguation is to determine which of the senses of an ambiguous word is invoked in a particular use of the word [3]. Starting from the algorithm of Yarowsky [5, 4] and the Naive Bayes Classifier (NBC) algorithm , in this paper we propose an original algorithm which combines their elements. This algorithm preserve the advantage of principles of Yarowsky ( *one sense per discourse and one sense per collocation*) with the known high performance of the NBC algorithm. Moreover, an agent is constructed accomplishing this algorithm.

## 1. Introduction

The word sense disambiguation ( WSD) is probably one of the most important open problem and it has now already a long "history" in computational linguistics [2, 1]. WSD problem has direct applications in some fields of text understanding as *information retrieval*, *text summarization*, *machine translation*.

The problem that arises in word sense disambiguation (WSD) in natural language is that many words (called polysemic), have several meanings or senses. These senses depend on the context they occur in. The task of disambiguation is to determine which of the senses of an ambiguous word is invoked in a particular use of the word [3]. Whenever a system's actions depend on the meaning of the text being processed, WSD is necessary.

The algorithms used in WSD are classified considering whether they involve supervised or unsupervised learning. Unsupervised learning can be viewed as clustering task while supervised learning is usually seen as a classification task. Dictionary based disambiguation, which we will present in the following section can be considered as intermediary between supervised and unsupervised disambiguation [3, 6].

---

2000 *Mathematics Subject Classification.* 68T50, 68Q32.

1998 *CR Categories and Descriptors.* I.2.7[**Computing Methodologies**]: Artificial Intelligence – *Natural Learning Processing*; 6.3[**Mathematicsof Computing**]: Statistical Computing – .

## 2. Dictionary based disambiguation

If we have no information about the senses of a target word $w$ we can follow disambiguation methods that rely on the definitions in dictionaries.

Notational conventions used in the following are:

- $w$– the word to be disambigued ( *target word*);
- $s_1, \cdots, s_K$— possible senses for $w$;
- $c_1, \cdots, c_I$—contexts of $w$ in corpus;
- $v_1, \cdots, v_J$— words used as contextual features for disambiguation of $w$.

Regarding of $v_1, \cdots, v_J$ there are two possibilities: they are colocates or co-occurrences with $w$. In the first case the contextual features occur in a fixed position near $w$, in a *window* of fixed length, centered on $w$. In the second case the contextual features occur together with $w$, in arbitrarily positions. We will consider the first sense of contextual features.

In [5] (1995), Yarowsky observed that there are constraints between different occurrences of contextual features that can be used for disambiguation. Two such constraints are:

- *One sense per discourse*: the sense of a target word is highly consistent within a given discourse (document);
- *One sense per collocation*: the contextual features (nearby words) provide strong clues to the sense of a target word.

For example, regarding the first constraint for the word *plant*, if his sense is in a first occurrence "living being", then later occurrences are likely to refer to "living beings" too. As the second constraint for *plant*, if the word *animal* occurs together with *plant*, this word is likely to be a clue word for the "living beings" sense.

The algorithm proposed by Yarowsky combines both constraints. It iterates building two sets , $F_k$ and $E_k$ for each sense $s_k$: $F_k$ contains characteristic collocations, $E_k$ is the set of contexts of the target word $w$ which are assigned to the sense $s_k$. The algorithm is as bellow [3]. Let us remark that a multi-set is denoted by $\{\{\cdots\}\}$:

*Initialization*
**for** *all sense* $s_k$ *of* $w$ **do**
$F_k = \{$*the set of collocations in definition from dictionary of* $s_k$ *sense of* $w\}$
**for** *all* $s_k$ *of* w **do**
$E_k = \Phi$

*One sense per collocation*

**While** *at least one* $E_k$ *changed in the last iteration* **do**
**for** *all sense* $s_k$ *of* $w$ **do**
$E_k = \{\{c_i \mid c_i \cap F_k \neq \Phi\}\}$
**for** *all sense* $s_k$ *of* $w$ **do**
$F_k = \{f_m \mid \forall n \neq k,\ \frac{P(s_k|f_m)}{P(s_n|f_m)} > \alpha\, (usually\ \alpha = 1)\}$

*One sense per discourse*
**for** *all document* $d_m$ (*context or set of contexts*)**do**
*determine the majority sense* $s_k$ *of* $w$ *in* $d_m$

*assign all occurrences of* $w$ *in* $d_m$ *to* $s_k$

## 3. Supervised disambiguation by Naive Bayes Classifier algorithm

In supervised disambiguation a tagged corpus or a semantic annotated corpus is available. Such annotated corpus is used in on-line product Senseval. The task in this case is to build a classifier which correct classifies a new context based on the contextual features occurring in this context. The classifier does no feature selection, but it combines the participation of all contextual features.

What a Naive Bayes Classifier realizes is the calculus of the sense $s'$ which for the target word $w$ and a given context $c$ satisfies the relation:

$$s' = argmax_{s_k} P(s_k \mid c) = argmax_{s_k} \frac{P(c \mid s_k)}{P(c)} P(s_k) \tag{1}$$

$$= argmax_{s_k} P(c \mid s_k) P(s_k).$$

The same value for $s'$ is obtained if we consider the logarithm of expression:

$$s' = argmax_{s_k} (logP(c \mid s_k) + logP(s_k)) \tag{2}$$

The Naive Bayes assumption is that the contextual features are all conditional independent:

$$P(c \mid s_k) = P(\{v_j \mid v_j \in c\} \mid s_k) = \prod_{v_j \in c} P(v_j \mid s_k). \tag{3}$$

Here $v_j$ represents any word in the context $c$.
This assumption has two consequences:

- the structure and order of words in context is ignored;
- the presence of one word in the context does not depend on the presence of another.

This is clearly not true, but there is a large number of cases in which the algorithm works well.

As regarding the probabilities $P(v_j \mid s_k)$ and $P(s_k)$, these are calculated from the labeled (annotated) corpus:

$$P(v_j \mid s_k) = \frac{C(v_j, s_k)}{C(s_k)} \quad P(s_k) = \frac{C(s_k)}{C(w)} \tag{4}$$

where $C(v_j, s_k)$ is the number of occurrences of $v_j$ in the contexts annotated with the sense $s_k$, $C(s_k)$ is the number of contexts with the sense $s_k$ and $C(w)$ is the total number of occurrences of the word $w$.

The NBC algorithm is:

*Training:*
**for** all senses $s_k$ of $w$ **do**
**for** all words $v_j$ in corpus) **do**
$P(v_j \mid s_k) = \frac{C(v_j, s_k)}{C(s_k)}$
**for** all senses $s_k$ of $w$ **do**
$P(s_k) = \frac{C(s_k)}{C(w)}$

*Disambiguation:*
**for** all senses $s_k$ of $w$ **do**
$score(s_k) = logP(s_k) + \sum_{v_j \in c} logP(v_j \mid s_k)$
*Calculate* $s' = argmax_{s_k} score(s_k)$

In [3] is reported that a disambiguation system based on this algorithm is correct for about 90 percents of cases.

## 4. A bootstrapping algorithm on the base of the principles: one sense per discourse and one sense per collocation

The algorithm begins by identifying a small number of training contexts. This could be accomplished by hand tagging with senses the contexts of $w$ for which the sense of $w$ is clear because some *seed collocations* [5] occour in these contexts.

This tagging is made on the base of dictionaries or by using the known online dictionary of senses WordNet . This initial set of annotated contexts is used for learning an *naive bayesian classifier*. This NBC will help in annotating new contexts. By repeating the process, the annotated part of corpus grows. We will

stop when the remaining unannotated corpus is empty or any new context can't be annotated.

The notational conventions are as above:

- $w$ is the polysemic word
- $S = \{s_1, s_2, \cdots, s_K\}$ are possible senses for $w$, as in a dictionary, or as obtained with WordNet.
- $C = \{c_1, c_2, \cdots c_I\}$ are contexts (windows) for $w$, as obtained for $w$ with an on-line corpus tool ( for example Cobuild). each $c_i$ is of the form:

$$c_i = w_1, w_2, \cdots, w_t, w, w_{t+1}, \cdots, w_z$$

where $w_1, w_2, \cdots, w_t, w_{t+1}, \cdots, w_z$ are words from the set $v_1, \cdots, v_J$ and $t$ and $z$ ( usually $z = 2t$) are selected by user.

Let us consider that the words $V = \{v^1, \cdots, v^l\} \subset \{v_1, \cdots, v_J\}$, where $l$ is small ( for example 2 ) are *surely* associated with the senses for $w$, such that the occurrence of $v^i$ in the context of $w$ determines the choice for $w$ of a sense (one sense per collocation).

For example, for the word *plant*, the occurrence in the same context of the word *life* means a sense ( let say A) , while the occurrence in the same context of the word *manufacturing* means another sense ( let say B). These rules can be done as a decision list:

$$if\ v^i\ occurs\ in\ a\ context\ of\ w\ (of\ z\ words) \Rightarrow\ s^i,\ i = 1, \cdots, l$$

So, some contexts can be solved from the set of contexts obtained as query results with Cobuild. Namely, we marked these contexts with A or B:

```
(A)industrial equipment and engineering plant.[p] The company
   insures
(A)hard currency. And so we've found a plant, and I have some
   seeds here from
(B)the planning and construction of the plant at Rabta near
   Tripoli and were
(A)A. japonica Japanese aucuba. A male plant, bearing panicles
   of purple-
(A)and experience of any individual plant in my garden alone
   is hardly
(A)aspect, features and animal and plant life."[p] [p] These
   were never
(A)in flower and it Is worth having a plant or two in the
   flower border or in
(B)all the allegations. It says the plant produces merely
   pharmaceuticals.
(A)or example, the issue of the role of plant respiration in
```

```
   a hydrological
(A)he USA announced in 1989 that 680 US plant species will
   be extinct in the wild
(B)d be looking at 75 to 100 jobs and a plant that would
   produce probably
(B)the Sellafield nuclear reprocessing plant. These are ‘cost
   plus" contracts
(B)[h] [p] SCIENTISTS have engineered a plant which could
   grow its own plastic
```

**Algorithm**

We start by defining a relation $\delta : WXC$, where $W$ is the set of words and $C$ is the set of contexts (set of array of words). If $w \in W$ is a word and $c \in C$ is a context, we say that $(w, c) \in \delta$ if exist a word $w1 \in c$ so that the words $w$ and $w1$ have the same root.

$C_{res} = \Phi$, determine the set $V = \{v^1, \cdots, v^l\}$
**For** *each context* $c$ *in* $C$ *apply the rules:*
**if** $(v^i, c) \in \delta, \Rightarrow$ *sense* $s'^i$, $i = 1, \cdots, l, C_{res} = C_{res} \cup \{c\}$
$C_{rest} = C \backslash C_{res}$
**While** $C_{rest} \neq \Phi$ **do** :
*Determine a set* $V^*$ *of words with maxim frequency in* $C_{res}$
*Define* $V = V \cup V^* = \bigcup_{j=1}^{l} V_{s_j}$,
*where* $V_{s_j}$ *is the set of words associate with the sense* $s_j$
*(***If** $v \in V^*$, *the context* $c$ *solved with the sense* $s_j$, *and* $(v, c) \in \delta$, *then* $v \in V_{s_j}$, *according with the principle "one sense per discourse")*
**For** *each* $c_i \in C_{rest}$ *apply the BNC algorithm* :

$$s_i^* = argmax_s P(s \mid c_i) = argmax_s \frac{P(c_i \mid s) \times P(s)}{P(c_i)} \tag{5}$$

$$= argmax_s P(c_i \mid s) \times P(s)$$

$$where\, P(c_i \mid s) = P(w_1 \mid s) \cdots P(w_t \mid s) P(w_{t+1} \mid s) \cdots P(w_z \mid s)$$

$$and P(w_i \mid s_j) = \begin{cases} 1 & if\ w_i \in V_{s_j} \\ \frac{nr.occ.w_i}{nr.\,total\,of\,words}) & else \end{cases}$$

$C_{res}^* = \{c_i \mid P(s_i^* \mid c_i) > N,\, N\, fixed\}$
$C_{res} = C_{res}^* \cup C_{res}$
$C_{rest} = C_{rest} \backslash C_{res}$

## 5. The application

5.1. **The Agent for words' disambiguation. General presentation.** The application is written in Visual C++ 6.0 (Figure 1) and implements the behavior

of an Intelligent Agent, whose purpose is to find the correct sense for a given word (the target word) in some given contexts (the word's disambiguation), using the algorithm described in the previous section.

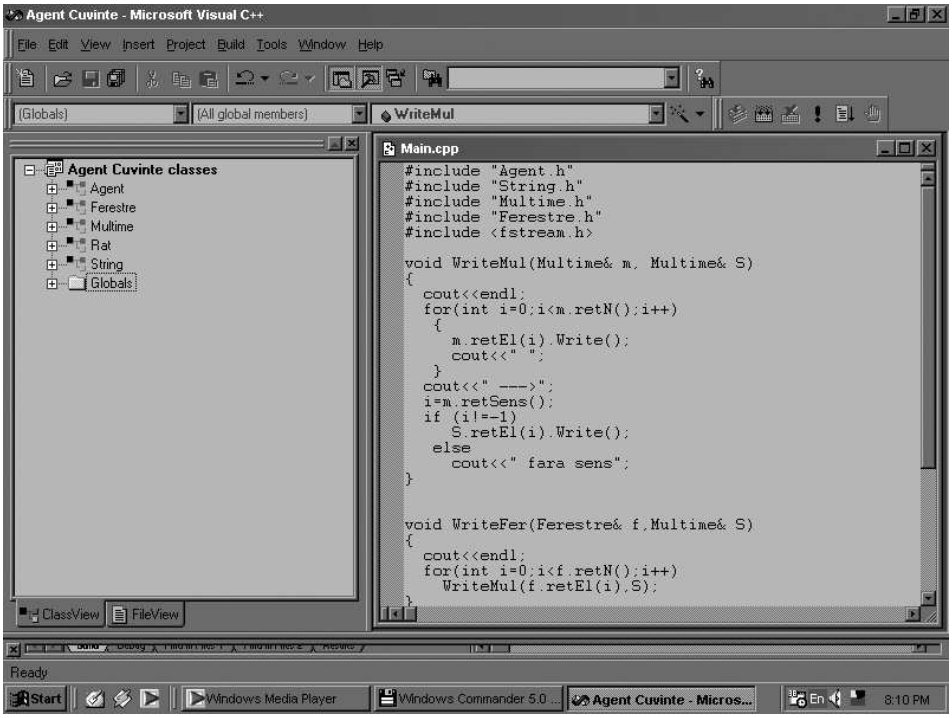


FIGURE 1. The Agent

The environment of this agent consists in some information which the agent reads from an input text file "in.txt":

- the target word($w$);
- the possible senses for $w$;
- the contexts for $w$;
- the words used as contextual attributes for $w$'s disambiguation.

On the basis of his environment, using the algorithm described in the previous section, the agent learns to find the correct sense of the target word in the given contexts.

5.2. **The Agent's design.** The basis classes used for implementing the agent's behavior are the following:

- **String**: defines the type *String* (array of characters), having methods for:
  - adding a char in a *String*;
  - accessing the length and the characters of a *String*;
  - displaying, comparing, concatenating *Strings*.
- **Set**: defines the type *Array* of strings (corresponding to a context which contains the target word $w$), associated with a sense of $w$. The main methods of this class are for:
  - adding a String in an *Array*;
  - accessing the number of elements and the strings of an *Array*;
  - testing the membership of a string in the *Array*;
  - setting the corresponding sense for $w$;
  - finding the reunion of two *Arrays*.
- **Contexts**: defines the type *Set* of arrays of strings (array of contexts), representing the contexts for which we want to associate a sense corresponding to $w$. The main methods of this class are for:
  - adding an element in the *Set*;
  - accessing the number of elements and the elements of a *Set*;
  - testing the membership of an array in the *Set*;
  - finding the difference of two *Sets*.
- **Agent**: the main class of the application, which implements the agent behavior and the learning algorithm (Figure 2).

  The private member data of this class are:
  - **Q**: the target word;
  - **S**: the set of senses for the target word;
  - **v**: the set of words used as contextual attributes for $Q$'s disambiguation;
  - **C**: the contexts for the target word.

  The public methods of the agent are the followings:
  - **readEnvironment**: reads the information about the environment from an input stream ;
  - **disambiguation**: the main (learning)algorithm of the agent used to find the correct senses of the target word in the environment's contexts;
  - **retQ**: returns the target word ($Q$);
  - **retS**: returns the set of senses of the target word ($S$);
  - **retV**: returns the member data $v$;
  - **retC**: returns the contexts for the target word ($C$);

  Besides the public methods, the agent has some private methods used in the method **disambiguation**.

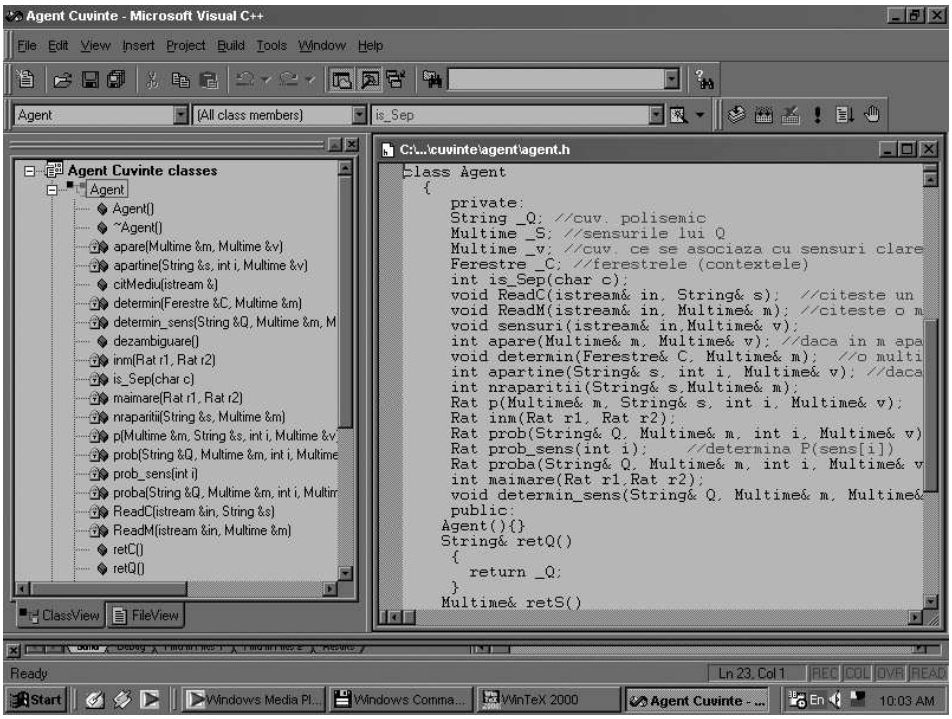


FIGURE 2. The main class of the Agent

5.3. **Experiment.** Using the application we accomplish the training of the agent in the following environment (given in the text file "in.txt"):

| | |
|---|---|
| **poarta** | the target word |
| **usa imbraca** | the senses of the target word |
| **clanta 1 se 2** | the words used as contextual attributes for $Q$'s disambiguation and the indexes of the corresponding sense of the target word |
| *poarta* **se deschide cu clanta** | the contexts of the target word |
| **se** *poarta* **rosu** | |
| **unde este** *poarta* | |
| **se stie ca cine** *poarta* **rosu este optimist** | |
| **daca** *poarta* **e inchisa nu intram** | |

After the agent reads the information from the environment, he applies the disambiguation algorithm for the given contexts. The result is shown below(each context is followed by the sense for the target word - found by the agent after the disambiguation).

---

*poarta* **se deschide cu clanta** — *usa*
**se** *poarta* **rosu** — *imbraca*
**unde este** *poarta* — *usa*
**se stie ca cine** *poarta* **rosu este optimist** — *imbraca*
**daca** *poarta* **e inchisa nu intram** — *usa*

---

We notice that if the Agent starts with a substantial initial knowledge (number of senses of the target word, set of words used as contextual attributes for the disambiguation) and if the environment consists in a big number of contexts, the the disambiguation (learning) algorithm works very well.

## References

[1] J.Allen : " Natural language understanding", Benjamin/Cummings Publ. , 2nd ed., 1995.
[2] D. Jurafsky, J. Martin: " Speech and language processing", Prentice Hall, 2000.
[3] C. Manning, H. Schutze: " Foundation of statistical natural language processing", MIT, 1999.
[4] P. Resnik, D. Yarowsky : " Distinguishing Systems and Distinguishing sense: new evaluation methods fot WSD ", Natural Language Engineering, 1 , nr 1, 1998.
[5] D. Yarowsky: "Hierarchical Decision Lists for WSD", Kluwer Acadmic Publishers, 1999.
[6] F. Sebastiani: " A tutorial on Automated Text Categorization", pp 1-25, ESSLLI 2001.
[7] D. Tatar: "Inteligenta artificiala: demonstrare automata de teoreme, prelucrarea limbajului natural", Editura Microinformatica, 2001.

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

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLVI**, Number 2, 2001

# MAHMUT PARLAR, "INTERACTIVE OOPERATIONS RESEARCH WITH MAPLE. METHODS AND MODELS", BIRKHÄUSER, BOSTON, 2000, ISBN 0-8176-4165-3

BAZIL PÂRV

The book is structured in 10 chapters, followed by a list of references and an index, all covering 468 pages. The first two chapters introduce the main topics referring to Operations Research (OR) and Maple, while the third, Maple and Mathematical Foundations of Operations Research, presents the main functionality of Maple in performing mathematical operations, with focus on those needed in OR field: algebra, calculus, linear algebra, differential equations, transform methods, and probability theory.

The next seven chapters are dedicated to a particular OR topic: linear programming, nonlinear programming, dynamic programming, stochastic processes, inventory models, queueing systems, and simulation. Every chapter follows the same pattern: an introduction, followed by the presentation of OR methods, techniques, and example problems, accompanied by solutions using Maple. A summary and a list of proposed exercises are given at the end of each chapter.

The linear programming chapter presents the general form of linear programming problem, the graphical and simplex solutions, exemplified for production and transportation problem, and two-person zero-sum games. Special cases, difficulties, sensitivity analysis, and duality are also covered.

The nonlinear programming chapter introduces the general nonlinear programming problem, convexity sets and functions with examples, unconstrained and inequality/equality constrained optimization, and Lagrangian duality.

The dynamic programming chapter starts with the presentation of the general concepts of dynamic programming problem, followed by discussion of some particular cases like models with a linear system and quadratic cost and continuous-time dynamic programming. Examples include the stagecoach problem, the infinite-stage problem, the constrained work force planning problem, the gambling model with myopic optimal policy, and optimal stopping problems.

Exponential distribution and Poisson processes, renewal theory, discrete-time Markov chains, and continuous-time Markov chains are the main topics discussed in the stochastic processes chapter. Accompanying examples include time-dependent

arrival rate in a restaurant, random walk, periodic-review inventory systems, machine maintenance problem, birth and death processes, a self-service car wash and vacuum facility with extra capacity, and response areas for emergency units.

The inventory models chapter introduces and classifies those models and the associated costs followed by a detailed presentation of deterministic and probabilistic models. Deterministic inventory models and related topics discussed are: the basic EOQ (economic order quantity) model, the EOQ model with backorders, the analysis of implied backorder costs, and quantity discounts. In the class of probabilistic inventory models are considered the continuous-review model (in approximate and exact formulations), the one-period model with random demand, and dynamic inventory models.

Markovian queueing systems, their transient solutions, queueing networks, and optimization of queueing systems are the main topics of the queueing systems chapter. Concrete problems and models discussed include birth and death processes, M/M/1, M/M/1/K, M/M/c, $M^X/M/1$, and $M/M/\infty$ queueing systems, serial queue with blocking, Jackson networks, transportation queueing processes, and optimal dynamic service rate control.

The last chapter is dedicated to simulation. It deals with methods for generating (pseudo-) random numbers (mixed-congruential, Maple's uniform random number generator, Kolmogorov-Smirnov test for uniformity), generating random variates from other distributions (exponential random variates, Maple's random variates generator), Monte Carlo simulation (used to evaluate definite integrals numerically and to simulate a static single-period problem), dynamic simulation models (inventory system with random yield, non-Markovian queues), and optimization of random search.

The book can be considered as an essential reference material for students, professors, researchers, and practitioners who learn, teach, and use OR principles, methods, and techniques in various areas, from management science, engineering, and mathematics departments in universities to R&D, marketing, production, and other departments in companies. Examples presented throughout the book illustrate the capabilities and usefulness of Maple computer algebra system in formulating and solving various real-world OR problems.

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

*E-mail address*: bparv@cs.ubbcluj.ro