Anul XLVIII 2003

S T U D I A
UNIVERSITATIS BABEŞ-BOLYAI

INFORMATICA

1

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

SUMAR – CONTENTS – SOMMAIRE

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLVIII**, Number 1, 2003

A NEW REINFORCEMENT LEARNING ALGORITHM

GABRIELA ŞERBAN

Abstract. The field of Reinforcement Learning, a sub-field of machine learning, represents an important direction for research in Artificial Intelligence, the way for improving an agent's behavior, given a certain feed-back about its performance. In this paper we propose an original algorithm (URU - Utility-Reward-Utility), which is a temporal difference reinforcement learning algorithm. Moreover, we design an Agent for solving a path-finding problem (searching a maze), using the URU algorithm.
Keywords: Reinforcement Learning, Intelligent Agents.

1. Reinforcement Learning

Reinforcement Learning (RL) is the way of improving the behavior of an agent, given a certain feedback about his performance.

Reinforcement Learning [3] is an approach to machine intelligence that combines two disciplines to successfully solve problems that neither discipline can address individually. Dynamic Programming is a field of mathematics that has traditionally been used to solve problems of optimization and control. However, traditional dynamic programming is limited in the size and complexity of the problems it can address.

Supervised learning is a general method for training a parameterized function approximator, such as a neural network, to represent functions. However, supervised learning requires sample input-output pairs from the function to be learned. In other words, supervised learning requires a set of questions with the right answers.

Unfortunately, there are many situations where we do not know the correct answers that supervised learning requires. For these reasons there has been much interest recently in a different approach known as reinforcement learning (RL). Reinforcement learning is not a type of neural network, nor is it an alternative to neural networks. Rather, it is an orthogonal approach that addresses a different, more difficult question. Reinforcement learning combines the fields of dynamic

Received by the editors: December 10, 2002.

2000 *Mathematics Subject Classification.* 68T05.

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

programming and supervised learning to yield powerful machine-learning systems. Reinforcement learning appeals to many researchers because of its generality. 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 is 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. It is the job of the RL system designer to define a reinforcement function that properly defines the goals of the RL agent. Although complex reinforcement functions can be defined, there are at least three noteworthy classes often used to construct reinforcement functions that properly define the desired goals;
- *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 exactly information about what he did well or wrong.

2. A Reinforcement Learning Problem

Let us consider the following problem:
The Problem Definition

We consider an environment represented as a space of states (each state is characterized by its position in the environment - two coordinates specifying the X-coordinate, respectively the Y-coordinate of the current position).

The goal of a robotic agent is to learn to move in the environment from an initial to a final state, on a shortest path (as number of transitions between states).

Notational conventions used in the followings are:

- $M = \{s_1, s_2, ..., s_n\}$ - the environment represented as a space of states;
- $si \in M$, $sf \in M$ - the initial, respectively the final state of the environment (the problem could be generalized for the case of the environments with a set of final states);
- $h : M \to P(M)$ - the transition function between the states, having the following signification: $h(i) = \{j_1, j_2, ..., j_k\}$, if, at a given moment, from the state i the agent could move in one of the states $j_1, j_2, ..., j_k$; we will call a state j that is accessible from state i ($j \in h(i)$) the *neighbor* (*successor*) state of i;
- the transition probabilities between a state i and each neighbor state j of i are the same, $P(i, j) = \frac{1}{card(h(i))}$ (we note with $card(M)$ the number of elements of the set M);

The Goal

The problem will consist in training the agent to find the shortest path to reach the final state sf starting from the initial state si.

For solving this problem, we propose in the followings a reinforcement learning algorithm, based on learning the states' utilities (values), in which the agent receives rewards from interactions with the environment.

3. The URU Algorithm (Utility-Reward-Utility)

The algorithm described in this section is an algorithm for learning the states' values, a variant of learning based on Temporal Differences [1].

The algorithm's idea is the following:

- the agent starts with some initial estimates of the state's utilities;
- during some training episodes, the agent will experiment some paths from si to sf (possible optimal), updating, properly, the states' utilities estimations;
- during the training process the states' utilities estimations converge to the exact values of the states' utilities, thus, at the end of the training process, the estimations will be in the vicinity of the exact values.

We make the following notations:

- $U(i)$ - the estimated utility of the state i;
- $R(i)$ - the reward received by the agent in the state i.

The URU Algorithm

The algorithm is shown in Figure 1.

---

(1) Initialize the state utilities with some initial values;
(2) Initialize the current state with the initial state $sc := si$;
(3) Choose a state s neighbor of sc ($s \in h(sc)$), using some known action selection mechanisms (ϵ-Greedy or SoftMax [2]), following the steps:
  (a) determine the set of successors of the current state ($m = h(sc)$);
  (b) if the current state has no successors (m is empty), return to the previous state ($s := sc$); otherwise go to step (c);
  (c) select from m a subset $m1$ containing the states that were not visited yet in the current training sequence;
  (d) choose a state s from $m1$ using a selection mechanism.
(4) determine the reward r received by the agent in the state sc;
(5) if the current state is not final, then update the utility of the current state as follows:

$$U(sc) := U(sc) + \alpha \cdot (r + \gamma \cdot U(s) - U(sc)) \tag{1}$$

where $\alpha \in (0, 1)$ is a fixed parameter (the *learning rate*), and $\gamma \in (0, 1)$ is a fixed parameter (the *reward factor*).
(6) $sc := s$;
(7) repeat the step 3 until sc is the final state;
(8) repeat the steps 2-7 for a given number of training episodes.

FIGURE 1. The Reward-Utility-Reward (URU) Algorithm.

We have to make the following specifications:

- the training process during an episode has the complexity in the worst case $O(n^2)$, where n is the number of the environment's states;
- in a training sequence, the agent updates the utility of the current state using only the selected successor state, not all the successors (the temporal difference characteristic).

4. Case Study

It is known that the estimated utility of a state [1] in a reinforcement learning process is the estimated *reward-to-go* of the state (the sum of rewards received from the given state to a final state). So, after a reinforcement learning process, the agent learns to execute those transitions that maximize the sum of rewards received on a path from the initial to a final state.

If we consider the reward function as: $r(s) = -1$ if $s \neq sf$, and $r(s) = 0$, otherwise, it is obvious that the goal of the learning process is to minimize the

number of transitions from the initial to the final state (the agent learns to move on the shortest path).

For illustrating the convergence of the algorithm, we will consider that the problem is one of learning the shortest path (the reward function is as we described above), and the environment has the following characteristics:

- the environment has a rectangular form;
- at a given moment, from a given state, the agent could move in four directions: North, South, East, West.

In the followings, we will enounce an original theorem that gives the conditions for convergence of the URU algorithm.

Theorem 1. *Let us consider the learning problem described above and which satisfies the conditions:*

- *the initial values of the states' utilities (the step (1) of the URU algorithm described in Figure 1) are calculated as:* $U(s) = -d(s, sf) - 2$*, for all* $s \in M$*, where* $d(s1, s2)$ *represent the Manhattan distance between the two states;*
- $\gamma \leq \frac{1}{3}$

In this case, the URU algorithm is convergent (the states' utilities are convergent after the training sequence).

The Theorem 1 proving is based on the following lemmas:

Lemma 2. *At the n-th training episode of the agent the following inequalities hold:* $U_n(i) \leq -2$*, for all* $i \in M$*.*

Lemma 3. *At the n-th training episode of the agent the following inequalities hold:* $|U_n(i) - U_n(j)| \leq 1$*, for each transition from* $i (i \neq sf)$ *to* j *made by the agent in the current training sequence.*

Lemma 4. *The inequalities* $U_{n+1}(i) \geq U_n(i)$ *hold for all* $i \in M$ *and for all* $n \in N$*, in other words the states' utilities increase from a training episode to another.*

Theorem 2 gives the equilibrium equation of the states' utilities after applying the URU algorithm.

Theorem 5. *In our learning problem, the equilibrium equation of the states' utilities is given by the following equation:*

$$U^*_{URU}(i) = R(i) + \frac{\gamma}{card(h(i))} \cdot \sum_{j \text{ successor of } i} U^*_{URU}(j) \tag{2}$$

for all $i \in M$*, where* $U^*_{URU}(i)$ *represents the exact utility of the state i, obtained after applying the URU algorithm. We note by* $card(M)$ *the number of elements of the set* M*.*

5. An Agent for Maze Searching

5.1. **General Presentation.** The application is written in JDK 1.4 and implements the behavior of an Intelligent Agent (a robotic agent), whose purpose is coming out from a maze on the shortest path, using the algorithm described in the previous section (URU).

We assume that:

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

5.2. **The Agent's Design.** The basis classes used for implementing the agent's behavior are the followings:

- **CState**: defines the structure of a *State* from the environment. This class has methods for:
  - setting components (the current position on the maze, the value of a state, the utility of a state);
  - accessing components;
  - calculating the utility of a state;
  - verifying if the state is accessible (does this contain or not an obstacle).
- **CList**: defines the structure of a list of objects. The main methods of the class are for:
  - adding elements;
  - accessing elements;
  - updating elements.
- **CEnvironment**: defines the structure of the agent's environment (it depends on the concrete problem - in our example the environment is a rectangular maze).
- **CNeighborhood**: the class that defines the accessibility relation between two states of the environment;
- **CRLAgent**: the main class of the application, which implements the agent's behavior and the learning algorithm.

  The private member data of this class are:
  - **m**: the agent's environment (is a *CEnvironment*);
  - **v**: the accessibility relation between the states (is a *CNeighborhood*);

  The public methods of the agent are the followings:
  - **readEnvironment**: reads the information about the environment from an input stream;

- **writeEnvironment**: writes the information about the environment in an output stream;
- **learning**: is the main method of the agent, which implements the URU algorithm; based on this algorithm, the agent updates the utilities of the environment's states.
- **next**: the agent determines the next state where to move (this is made after the learning process took place).

Besides the public methods, the agent has some private methods used in the method **learning**.

5.3. **Experimental Results.** For our experiment, we consider the environment shown in Figure 2. The state marked with 1 represents the initial state of the agent, the state marked with 2 represents the final state and the states filled with black are containing obstacles (which the agent should avoid).

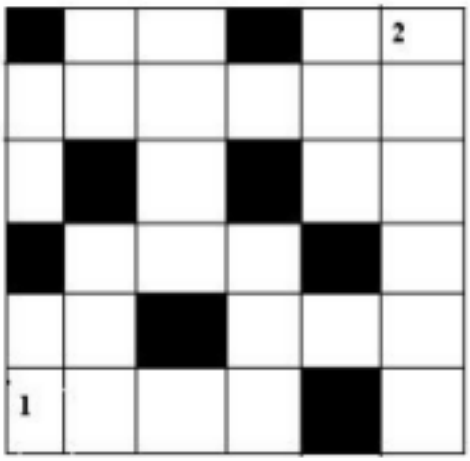


FIGURE 2. The agent's environment

For the environment described in Figure 2, we use the URU algorithm, with the following initial settings:

- $\gamma = 0.9$;
- $\alpha = 0.01$;
- number of episodes = 10;
- as a selection mechanism, we choose the ϵ-Greedy selection, with ϵ=0.1.

The results obtained after the URU learning are presented in Table 1. The states from the environment are numbered from 1 to 36, starting with the corner

TABLE 1. The states' utilities after the training episodes with the URU algorithm

State	Episode 1	Episode 2	Episode 3	Episode 4	Episode 5	Episode 6	Episode 7	Episode 8	Episode 9	Episode 10
1	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
2	-6.0000	-6.0000	-6.0000	-6.0000	-6.0000	-6.0000	-5.9710	-5.9422	-5.9422	-5.9422
3	-5.0000	-5.0000	-5.0000	-5.0000	-5.0000	-5.0000	-4.9780	-4.9561	-4.9561	-4.9561
4	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
5	-3.0000	-2.9760	-2.9760	-2.9760	-2.9522	-2.9522	-2.9522	-2.9522	-2.9522	-2.9522
6	-2.0000	-2.0000	-2.0000	-2.0000	-2.0000	-2.0000	-2.0000	-2.0000	-2.0000	-2.0000
7	-8.0000	-8.0000	-8.0000	-8.0000	-8.0000	-8.0000	-7.4610	-7.1124	-7.1124	-7.1124
8	-7.0000	-7.0000	-7.0000	-7.0000	-7.0000	-7.0000	-6.9640	-6.9267	-6.9267	-6.9267
9	-6.0000	-6.0000	-6.0000	-6.0000	-6.0000	-6.0000	-5.9650	-5.9303	-5.9303	-5.9303
10	-5.0000	-5.0000	-5.0000	-5.0000	-5.0000	-5.0000	-5.0000	-4.9779	-4.9779	-4.9779
11	-4.0000	-3.9790	-3.9790	-3.9790	-3.9581	-3.9581	-3.9581	-3.9436	-3.9436	-3.9436
12	-3.0000	-3.0000	-3.0000	-3.0000	-2.9919	-2.9919	-2.9919	-2.9839	-2.9839	-2.9601
13	-9.0000	-9.0000	-9.0000	-9.0000	-9.0000	-9.0000	-8.3031	-7.8600	-7.8600	-7.8600
14	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
15	-7.0000	-7.0000	-7.0000	-7.0000	-7.0000	-7.0000	-6.9580	-6.9580	-6.9580	-6.9580
16	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
17	-5.0000	-4.9720	-4.9720	-4.9720	-4.9720	-4.9720	-4.9720	-4.9720	-4.9720	-4.9720
18	-4.0000	-3.9850	-3.9850	-3.9850	-3.9641	-3.9641	-3.9641	-3.9435	-3.9435	-3.9230
19	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
20	-8.3664	-8.3166	-7.7394	-7.7394	-7.6956	-7.6956	-7.6521	-7.6521	-7.0875	-7.0498
21	-7.9510	-7.9024	-7.8541	-7.8541	-7.8063	-7.8063	-7.7592	-7.7592	-7.7122	-7.6655
22	-6.9640	-6.9282	-6.8926	-6.8926	-6.8573	-6.8573	-6.8573	-6.8573	-6.8218	-6.7866
23	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
24	-5.0000	-4.9720	-4.9720	-4.9720	-4.9442	-4.9442	-4.9442	-4.9167	-4.9167	-4.8893
25	-10.9300	-10.8584	-10.7872	-9.7732	-9.7732	-9.7105	-9.7105	-9.6483	-9.5865	-9.5233
26	-9.2171	-9.1601	-8.4560	-8.4138	-8.3629	-8.3214	-8.2712	-8.2303	-7.5718	-7.5273
27	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
28	-7.9510	-7.9024	-7.8541	-7.8541	-7.8063	-7.7588	-7.7588	-7.7117	-7.6691	-7.6268
29	-6.9640	-6.9282	-6.8927	-6.8927	-6.8573	-6.8166	-6.8166	-6.7761	-6.7761	-6.7359
30	-6.0000	-5.9650	-5.9650	-5.9650	-5.9303	-5.9017	-5.9017	-5.8675	-5.8675	-5.8058
31	-11.9230	-11.8466	-11.7707	-10.5639	-10.5006	-10.4349	-10.3725	-10.3079	-10.2438	-10.1801
32	-10.9298	-10.9298	-10.8578	-10.7846	-10.7120	-10.6445	-10.5731	-10.5047	-10.4342	-10.4342
33	-9.9430	-9.9430	-9.8864	-9.8864	-9.8864	-9.1123	-9.1123	-9.0559	-9.0068	-9.0068
34	-8.9500	-8.9500	-8.9003	-8.9003	-8.9003	-8.2313	-8.2313	-8.1823	-8.1376	-8.0896
35	-7.9570	-7.9084	-7.8662	-7.8662	-7.8184	-7.7768	-7.7768	-7.7768	-7.7768	-7.7293
36	-7.0000	-6.9580	-6.9580	-6.9580	-6.9163	-6.8806	-6.8806	-6.8806	-6.8806	-6.8393

left-up, in order of the lines. On the columns are presented the estimated values of the states' utilities, during the training episodes.

From Table 1 it is obvious that the states' utilities grow during the training episodes. After the training, the agent will report the learned path (from the initial to the final state), that is the path **(6-1), (5-1), (5-2), (4-2), (4-3), (4-4), (5-4), (5-5), (5-6), (4-6), (3-6), (2-6), (1-6)**. As a policy for moving in the environment after the learning, we consider that, from a given state, the agent will move to a neighboring state that was not visited yet, and having a maximum utility (of course, to determine the policy, we could use some probabilistic action selection mechanisms).

In order to illustrate the experimental results, we will give, in the followings, graphical representations that confirm the theoretical results from the previous sections. Figure 3 presents the change of the initial state's utilities during the training episodes (it is obvious that the utilities grow during the training).

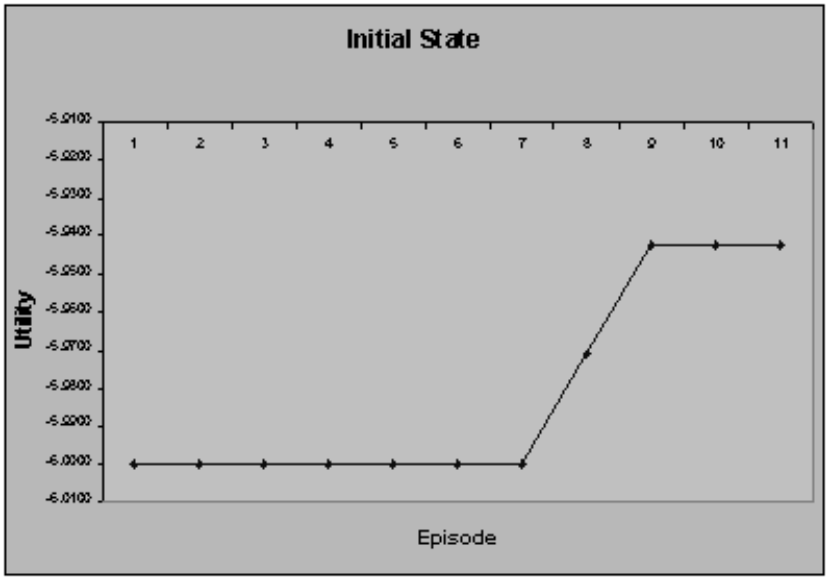


FIGURE 3. The initial state's utilities during the training process

Figure 4 presents the graphical representation of the states' utilities on the first training episode, and Figure 5 presents the graphical representation of the states' utilities on the last training episode. Analyzing comparatively the two figures, we observe that, for each state of the environment, the utilities grow during the training.

In Figure 6 we present comparatively the states' utilities on the first, the 10th and the 5th training episode.

5.4. **Experimental comparison between the URU algorithm and the TD [1] (Temporal Difference) algorithm.** In section 4 we illustrate that, in the case of the learning problem in the environment shown in Figure 2, by applying the TD algorithm the states' utilities are not convergent. This fact is presented in Table 2, where the utilities of the first five states during the training are described, results obtained by applying the TD algorithm for our problem.

From Table 2 it is obvious that the states' utilities have not a monotonic behavior, in other words, for some states the utilities increase, for other states the utilities decrease along the training, which does not guarantee the convergence of the algorithm.

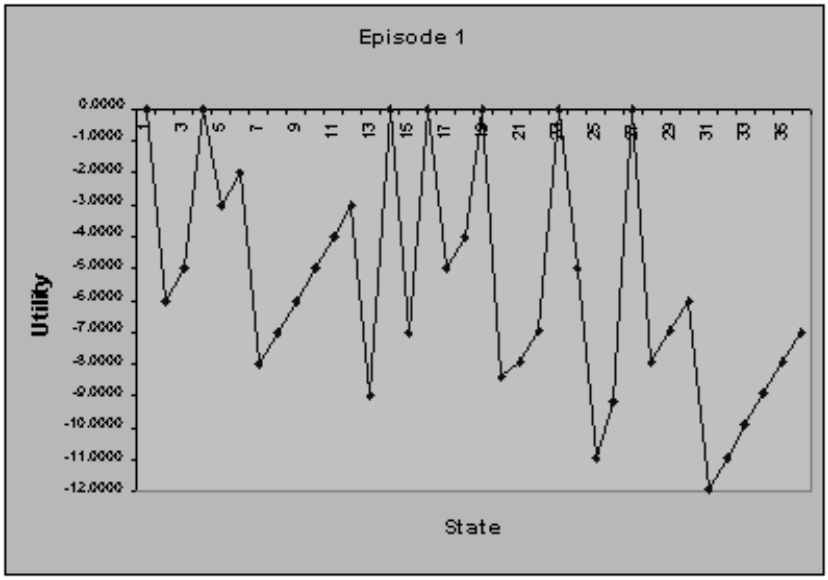


FIGURE 4. The states' utilities on the first training episode

TABLE 2. The states' utilities after the training episodes with the URU algorithm

State	Episode 1	Episode 2	Episode 3	Episode 4	Episode 5	Episode 6	Episode 7	Episode 8	Episode 9	Episode 10
1	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
2	-6.0000	-6.0000	-6.0000	-6.0002	-6.0002	-6.0002	-6.0002	-6.0002	-6.0202	-6.0202
3	-5.0000	-5.0000	-5.0000	-5.0200	-5.0200	-5.0400	-5.0400	-5.0400	-5.0600	-5.0600
4	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
5	-2.9900	-2.9801	-2.9703	-2.9703	-2.9703	-2.9606	-2.9606	-2.9606	-2.9510	-2.9510
...										

6. Conclusions and Further Work

As we have mentioned in the previous sections, the URU algorithm is a variant of a RL algorithm based on temporal differences (if $\gamma = 1$ URU becomes the classical temporal difference learning algorithm - TD).

In comparison with the classical TD algorithm, the following remarks may be considered. These follow naturally from the theoretical results described in Section 4):

- the states' utilities grow faster in the URU algorithm than in TD algorithm, in other words $U_{URU}(i) > U_{TD}(i)$, for all $i \in M$, which means

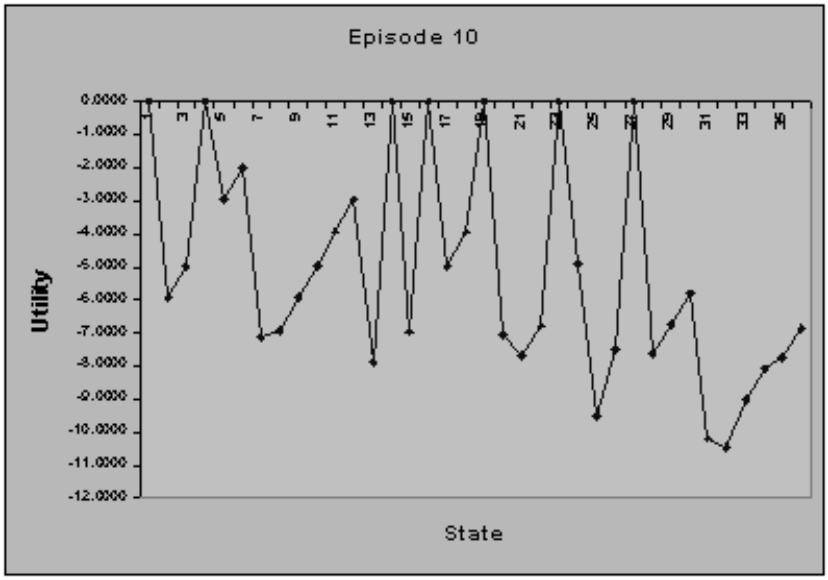


FIGURE 5. The states' utilities on the last training episode

that the URU algorithm converge faster to the solution than the TD algorithm;

- in the case of our learning problem, as we proved in Theorem 1, for $\gamma = 1$ (the TD algorithm), we cannot prove the convergence of the states' utilities.

Further work is planned to be done in the following directions:

- to analyze what happens if the transitions between states are nondeterministic (the environment is a Hidden Markov Model [4]);
- to analyze what happens if the reward factor (γ) is not a fixed parameter, but a function whose values depend on the current state of the agent.
- to develop the algorithm for solving path-finding problems with multiple agents.

REFERENCES

[1] Russell, S.J., Norvig, P.: Artificial Intelligence. A Modern Approach. Prentice-Hall, Englewood Cliffs, NJ, 1995

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

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

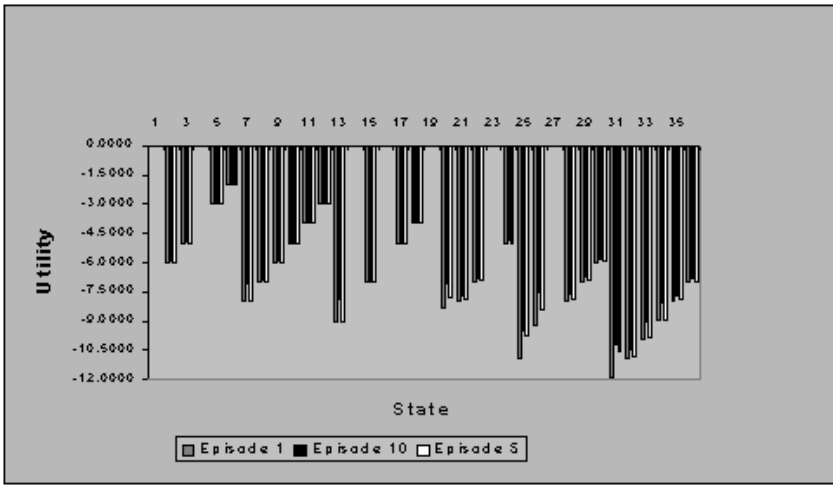


FIGURE 6. The states' utilities on the first, the 10th and the 5th training episode

[4] Serban, G.: Training Hidden Markov Models – A Method for Training Intelligent Agents, Proceedings of the Second International Workshop of Central and Eastern Europe on Multi-Agent Systems, Krakow, Poland, 2001, pp. 267-276

BABEŞ-BOLYAI UNIVERSITY, CLUJ-NAPOCA, ROMANIA
E-mail address: gabis@cs.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLVIII**, Number 1, 2003

SCHEDULING OPTIMALITY FOR THE PARALLEL EXECUTION OF LOGIC PROGRAMS

MONICA VANCEA AND ALEXANDRU VANCEA

Abstract. Logic programming is the most widespread programming paradigm used in artificial intelligence, a domain which needs intensive computing resources. Parallel execution of logic programs is the most effective speedup factor which can be applied for obtaining reasonable execution time in some cases. Parallelizing compilers have the task to exploit the inherent parallelism from the sequential programs having as the ultimate goal their efficient execution by means of building a time optimal schedule. These tools focus on the inherent parallelism available at the level of the logic languages operational semantics. Besides particular techniques for achieving optimal execution for specific classes of logic programs, one question arises naturally: given an arbitrary logic program and a machine model which assumes sufficient but finite resources, is it always possible to build a time optimal schedule ? This paper defines the notion of time optimality and proves that in the general case, no time optimal schedule can be built for a logic program, because there are classes of logic programs which require infinite resources for accepting time optimal schedules.

1. Preliminaries

Logic programming is the most popular programming model used in the area of artificial intelligence. When using huge knowledge based systems, acceptable run time for receiving the output results can be obtained only by implementing parallel execution among the tasks generated by the corresponding goals. So, parallel logic programming is sometimes the only possibility to deal with the complexity of some artificial intelligence problems and on the other hand can be an adequate solution for many significant run time optimizations even for the actual solutions [2, 4, 6, 10].

Parallelizing compilers try to automatically and efficiently exploit the parallelism available in a given program, transforming the programs into their parallel

Received by the editors: January 15, 2003.

2000 *Mathematics Subject Classification.* 68M20, 68N17, 68Q10.

1998 *CR Categories and Descriptors.* D.1.6. [**Software**]: Programming Techniques – *Logic Programming*; D.1.3. [**Software**]: Programming Techniques – *Parallel Programming*; D.2.9. [**Software**]: Software Engineering – *Cost Estimation*; D.4.1. [**Software**]: Operating Systems – *Scheduling*.

versions taking into account the existing data dependences [5, 7]. Naturally, this transformations must assure that the semantics are the same.

Definition. Two program codes are said to be ***semantic equivalent*** if one of them can be obtained from the other by applying a sequence of data dependence preserving transformations.

The vast majority of studies upon optimality assume that the machine model has ***sufficient but limited resources***, meaning that the architecture can run any program for which the number of resources needed is bounded by some arbitrary integer [1, 8, 9]. We will denote it further as R and we make some standard assumptions about the operations to be scheduled: for simplicity, we assume, without loss of generality, that *any resolution step of an elementary clause takes one machine cycle.*

Informally, through optimal parallelization of a program code, we understand obtaining a semantically equivalent (SE) version of it which manages at every moment t to schedule in parallel the execution of all its independent operations. That's why we will characterize such a program code as *time optimal.* Thus, the parallelization process is optimal if we obtain a SE time optimal program code.

Formally, we give below three alternative definitions of this concept.

Definition. A program code P is said to be ***time optimal*** if any of the following statements is true:

a) for every operation w executed at moment t, there exists a dependence chain of length t which ends at w;
b) every execution E of P is running in the shortest possible time with regard to the P's data dependences;
c) the length of any execution E (interpreted as an execution path in the data dependence graph of P) is the length of the longest data dependence chain from P.

2. Forward execution

When dealing with conditional predicates at the level of some loopings (recursive calls for example), static scheduling alone cannot assure processors workload balance, due to possible strongly different execution times required by the different branches of such a decision structure. In general, these tests can not be evaluated at compile time, so a scheduler has no information on which can decide a proper load balance for obtaining a reasonable efficiency.

That is why in branch intensive programs time optimality cannot be achieved without *forward execution* of branches, that is executing everything it can be executed (as the time optimality definition requires) on every branch in advance, independently of the results of decision testing. This assures that no processor will be idle and that after evaluating the decision, the results will already be there, computed, thus contributing to a significant speedup.

```
predicates
  p(integer,integer,integer).
  q(integer,integer,integer,integer).
clauses
  p(X_init,Y_init,0) :- !. % starting values for the iterative loop
  p(X,Y,N) :- N1 is N-1, p(X1,Y1,N1), q(X,Y,X1,Y1).
  q(X,Y,X1,Y1) :- X1>Y1, Z is f(X1), X is g(Z), Y is e(X). (A)
  q(X,Y,X1,Y1) :- X1<=Y1, X is h(X1), Y is e(X). (B)
```

FIGURE 1. Code sequence illustrating the use of forward execution

We can define forward execution as follows:

Definition. Let S be a clause which is resolution dependent on a test clause T in a logic program code. During the execution of the program code, the clause S is said to be ***forward executed*** if it will be scheduled before or concurrent with the resolution of T.

This means that the system will do useless work for obtaining better results. Thus, there will be execution histories for which S will execute but its result will not contribute to the program's final output in any way.

We illustrate below the potential benefits of applying forward execution for the component clauses of a looping (recursive) predicate which has conditional clauses. Let's consider the code sequence from Figure 1, which illustrates a conditional iterative process (the input-output flow patterns are $p(o, o, i)$ and $q(o, o, i, i)$ and f, g, h and e are numeric functions).

The sequential execution of the N calls requires between $3N$ (if all calls will choose the B branch) and $4N$ (if all calls will choose the A branch) machine cycles (remember that we assumed for simplicity that every elementary clause resolution takes one cycle). With forward execution $2N + 3$ cycles are needed (evident from table 1, where we reduced the 4 sequential iteration steps to a compact 2 resolution steps iteration), so for large N values the method will improve the runtime execution by a factor of 2. Obviously, the operations scheduled in a time step are executed in parallel. In Table 1 we illustrate what we can call the execution model of the looping predicate, in which a call requires two units of time.

Let's notice that, by forward execution, we compute $z := f(x_B)$ in advance on the false branch (variant B, even if maybe the next iteration won't take the true branch path so no z value will be needed) and together with the test evaluation we also compute in advance $x_A := g(z)$. However, if the test output is false we do not need this values at all. So, time optimality is achieved by adding an extra resources cost.

Time step	Scheduled operations		
	True branch	False branch	
1	test$(x > y)$; $z := f(x)$; $x_B := h(x)$;		
2	$x_A := g(z)$	$y := e(x_B)$	
3	$y := e(x_A)$; $z := f(x_A)$		
4, 6, ..., $2k$	$x_A := g(z)$	test$(x > y)$	$x_B := h(x_B)$
5, 7, ..., $2k+1$	$y := e(x_A)$; $z := f(x_A)$ Recursive call on the (A) branch (True)	$y := e(x_B)$; Recursive call on the (B) branch (False)	$z := f(x_B)$

TABLE 1. Time optimal schedule with forward execution for the code in Figure 1

Ideally, making abstraction of the real execution conditions, which may vary a lot from case to case, the amount of parallelism which can be exploited is limited only by the data dependences between the program's statements (this defines the so called ***inherent parallelism***). Hardware resource constraints, such as processors and memory, may be eliminated, at least in theory, because we always may add extra technical components to overcome the lack of resources. That is why the most widespread execution models assume as we mentioned *finite but unlimited resources.*

3. Time optimality for logic programs containing conditional clauses

We approached forward execution in section 2 because it is evident that for obtaining a time optimal schedule we have to apply it. Anyway, we will show that there are cases when a time optimal schedule cannot be built with finite resources. This can happen because of the conditionals which are part of the body of a recursive predicate, when one branch can prevent the forward execution of some activities belonging to the other branch. Then, it can be the case that time optimal scheduling (based on its definition) forces at some moment t the parallel execution of much more statements than the R resources can afford so we will conclude that no time optimal schedule can be built for general logic programs. Intuitively, we observe that conditional predicates combined with data dependence restrictions prevent the load balancing of the activities which have to be scheduled in parallel, by means of forward execution and obbeying the definition of time optimality. This makes the finite but unlimited R resources of our machine model to be insufficient for building a time optimal schedule in the general case. This is because the number of resources needed becomes a function of time t, an unbounded value, even if we accept that any program run on our machine model will eventually finish its execution. So, *general practical optimal scheduling is an intractable problem.*

```
predicates
  p(integer,integer,integer).
  q(integer,integer,integer,integer,integer,integer,integer).
clauses
  p(X_init,Y_init,Z_init,0) :- !. % start values for iterative loop
  p(X,Y,Z,N) :- N1 is N-1, p(X1,Y1,Z1,N1), q(X,Y,Z,X1,Y1,Z1,N1).
  q(X,Y,Z,X1,Y1,Z1,N1) :- Z1>Y1, Z is f(X1,N1), Y is e(X1,Z). (A)
  q(X,Y,Z,X1,Y1,Z1,N1) :- Z1<=Y1, X is h(X1), Y is e(X,Z1). (B)
```

FIGURE 2. Code sequence illustrating a conditional iterative process

We will ellaborate more formally on this intuitive observations in the following, taking a suitable example to illustrate our point of view.

Theorem 1. *A general logic program has no time optimal schedule.*

Proof. Let's notice that if we find only one class of logic programs and only one particular execution history for which no time optimal schedule can be built then our result holds. Recall that we assumed that any resolution step requires exactly one time unit and that the definition of a time optimal schedule requires any operation to be executed immediately after its input data become available (without any supplimentary resource access restriction). So, we can consider for example the program code in Figure 2.

There, the input-output flow patterns are $p(o, o, o, i)$ and $q(o, o, o, i, i, i, i)$. Predicate p is a iterative recursive predicate which associates at any iteration current values for the variables X, Y and Z by calling the q predicate. Current X, Y and Z values are computed using the X, Y and Z values from the previous iteration (these previous values are identified in the above code by variables $X1$, $Y1$ and $Z1$ and they are passed as parameters from one iteration to the next by means of the recursive call of the p predicate). We will identify some particular predicates from the above code as follows:

S_0 ≡ p(X,Y,Z,N); S_1 ≡ Z is f(X1,N1); S_2 ≡ X is h(X1);
S_3 ≡ Y is e(X1,Z); S_4 ≡ Y is e(X,Z1); T_1 ≡ Z1>Y1; T_2 ≡ Z1<=Y1;

Using these notations we can identify the following dependences:

S_0 → S_0 (recursive self-dependence for accomplishing the iterative process);
S_2 → S_4 (output X from S_2 is used as input in S_4);
S_1 → S_3 (output Z from S_1 is used as input in S_3);
S_2 → S_1 (X1 used as input in S_1 is in fact the output X from the S_2 previous iteration);
S_2 → S_3 (X1 used as input in S_3 is in fact the output X from the S_2 previous iteration);

$S_2 \rightarrow S_2$ (iteration carried self-dependence – X1 used as an input parameter for function h is in fact the output X from the previous iteration of the same clause);

$S_1 \rightarrow T_1$,T_2 (the output Z from S_1 is used as input in the tests T_1 and T_2);

S_3, $S_4 \rightarrow T_1$,T_2 (the output Y from S_3 or S_4 is used as input in the tests T_1 and T_2);

We will refer further to a particular execution history, namely the one that takes the (A) branch for the first n1 iterations (T_1 evaluated to *false* and T_2 evaluated to *true*) and the (B) branch for the remaining n2 ones (T_2 evaluated to *false* and T_1 evaluated to *true*), with n2 $\leqslant$ n1. So, $N = n1 + n2$. Taking into account the identified dependences and considering that the R resources of our machine model do not add any other execution restrictions, any time optimal execution of our program code will need $n1 + 3$ time units (remember that we assume that every execution step requires exactly one machine cycle – or time unit – and that the definition of a time optimal schedule requires that any operation takes place as soon as the inputs are available, with no resource constraints). This becomes evident looking at Table 2 where we show which operations are executing at each time step. It is easy to see the pattern for the first $n1$ iterations (recursive calls): for each k, $2 \leqslant k \leqslant n1$, iteration k executes the clause $x_k := h(x_{k-1})$ (S_2) at time step k and the clause $y_k := E(x_k, z)$ (S_4) together with the test T_2 at the time step $k + 1$.

We must notice that in the meantime no statement is available for the forward execution on the (A) branch, due to the fact that the first execution of S_1 must wait on the final value of X issued by the self dependent predicate S_2 after the n1 iterations on the (B) branch. This value for X will not change further at all (we said that the next $n2$ iterations all will go on the (A) branch). So, we have now the function f and the last value of X available (from the $n1$ time step), making theoretically possible that all the bindings to the Z variables to be made simultaneously (in the same time unit). This can be done if we apply scalar expansion [3] for being able to retain every instance of Z in a separate memory cell (remember that we have finite but sufficient resources). So, for a time optimal execution we must have at the time step $n1+1$, $n2+2$ operations: the $n2$ bindings for Z's together with the last test of the first $n1$ iterations (T_2) and the binding $y_{n1} := E(x_{n1}, z)$. After we have all the Z values, the same reasons force us to compute all the data dependent Y values of S_3 in the next time step. We can do this applying again scalar expansion for the Y values and knowing that we have enough resources for this. So, for a time optimal execution we must have at the time step $n1 + 2$, $n2 + 1$ operations: the $n2$ bindings for the Y instances and the evaluation of the test T_1. After that, the only remaining operations are the $n2$ tests T_1 which all can be evaluated in the time step $n1 + 3$, because the values being compared are now all available.

Time step	**Scheduled operations**		
1	$z > y_0$		$x_1 := h(x_0)$
2		$y_1 := E(x_1, z)$	$x_2 := h(x_1)$
3	$z > y_1$	$y_2 := E(x_2, z)$	$x_3 := h(x_2)$
4	$z > y_2$	$y_3 := E(x_3, z)$	$x_4 := h(x_3)$
. . .	. . .	. . .	. . .
$n2$	. . .	. . .	. . .
. . .	. . .	. . .	. . .
$n1-1$	$z > y_{n1-3}$	$y_{n1-2} := E(x_{n1-2}, z)$	$x_{n1-1} := h(x_{n1-2})$
$n1$	$z > y_{n1-2}$	$y_{n1-1} := E(x_{n1-1}, z)$	$x_{n1} := h(x_{n1-1})$; *iteration* $i = n1$
$n1+1$	$z > y_{n1-1}$	$y_{n1} := E(x_{n1}, z)$	$z_1 := f_1(x_{n1})$, $z_2 := f_2(x_{n1})$, $\ldots$, $z_{n2} := f_{n2}(x_{n1})$
$n1+2$	$z > y_{n1}$	$y_{n1+1} := E(x_{n1}, z_1) \ldots$	$\ldots y_{n1+n2} := E(x_{n1}, z_{n2})$
$n1+3$	$z > y_{n1+1}$	$z_2 > y_{n1+2} \ldots$	$\ldots z_{n2} > y_{n1+n2}$

TABLE 2. Time optimal schedule for the code sequence in Figure 2

So, our analysis reveals that any time optimal execution of the above program code will need $n1 + 3$ time units. Also, we need

- $n2 + 2$ resources in the $n1 + 1$ time step;
- $n2 + 1$ resources in the $n1 + 2$ time step;
- $n2$ resources in the $n1 + 3$ time step.

The problem in this case is that the number of resources (processors) needed at a time step is a function of that time step (Resources($n1+1$) $= n2+2 = N-n1+2$). But if we have $N >> R$ with $n1$, $n2 >> R$ also, this means that ***even with the finite but sufficient resources*** that we considered in our machine model (the largest assumption we can made anyway for practical purposes) ***we have not enough resources available to schedule the required operations for an optimal execution***. So, no optimal schedule exists for the execution of the above logic program code.

We conclude then, that in the general case, no time optimal schedule is guaranteed to be found for a given program considering ***finite*** resources. ■

4. Conclusions and future work

We showed in section 3 that a time optimal schedule cannot be built with finite resources for a general logic program. We informally characterized one class of logic programs (namely those containing conditional predicates) that has no time optimal schedule and explained why it is so. This was sufficient for proving

that no time optimal schedule exists for a logic program in the general case. A thorough analysis of logic program features which determine the conditions under time optimal schedules exists is what we intend to do as future research.

References

[1] **F.E.Allen**, ***Program optimization***, in *Annual Review in Automatic Programming 5, International Tracts in Computer Science and Technology and their Applications*, vol.13, Pergamon Press, Oxford, England, 1969, 239–307.

[2] **K.A.M. Ali, R. Karlsson**, ***OR-Parallel Speedups in a Knowledge Based System: on Muse and Aurora,*** in *FGCS'92*, Tokyo, 1992.

[3] **D.Bacon, S.Graham and O.Sharp**, ***Compiler Transformations for High-Performance Computing,*** in *ACM Computing Surveys*, vol.26, no.4, December 1994, 345–420.

[4] **R. Bahgat and S. Gregory**, ***Pandora: Non-deterministic Parallel Logic Programming***, in G.Levi and M.Martelli, editors, *Proc. of the 6th International Conference on Logic Porgramming.* MIT Press, 1990.

[5] **Utpal Banerjee**, ***Dependence Analysis***, Kluwer Academic Publishers, 1997.

[6] **K.L. Clark, S. Gregory**, ***PARLOG: Parallel Programming in Logic***, in *A.C.M. TOPLAS*, Vol. 8, No.1, Jan. 1986.

[7] **Grigor Moldovan, Alexandru Vancea, Monica Vancea**, ***Data dependence testing for automatic parallelization***, Studia Univ. Babeş-Bolyai, Informatica, vol.II, no.1, 1997, 3–18.

[8] **D. Padua and M. Wolfe**, ***Advanced compiler optimizations for supercomputers***, in *Communications of ACM*, 29, 1986, 1184–1201.

[9] **Alexandru Vancea and Monica Vancea**, ***Efficient Parallel Code Generation for nested for-loops*****,** Seminar on Computer Science, Preprint no.2, 1997, 179–188.

[10] **Monica Vancea**, ***Executia paralela a programelor logice***, Sesiunea de Comunicări Ştiinţifice *Economia României la orizontul anului 2000*, 14-15 noiembrie 1998, Cluj-Napoca, in *Studii şi Cercetări Economice*, vol. XXVIII–XXIX, 1988, 985–995.

Faculty of Economic Science, Babeş-Bolyai University, Cluj-Napoca, Romania
E-mail address: vancea@econ.ubbcluj.ro

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

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLVIII**, Number 1, 2003

WORDS CLUSTERING IN QUESTION ANSWERING SYSTEMS

DOINA TĂTAR AND GABRIELA ŞERBAN

Abstract. Clustering words can be useful in construction of a hierarchy of hypernyms or a set of synonyms for languages different of English for which doesn't exist such hierarchy as WordNet (as in Romanian language case). A such of hierarchy is very important in some problems of disambiguation [10], as to perform automatic query expansion in a QA system for Romanian [7]. In this paper we we describe how a list of similar words with a given word can be constructed. Some words and word-clusters similarity measures are discussed.

The experiments are made using a Romanian corpus.

1. Introduction

Semantic knowledge is increasingly important in NLP. The key of organizing semantic knowledge is to define reasonable similarity measures between words. The purpose to develop a hierarchy of words based on a untagged corpus can be realized by using hierarchical and non hierarchical clustering algorithms. In many papers the similarity between two words is obtained by the n-grams models [8], by mutual information [2] or by syntactic relations [9]. One other mode to define this similarity is the vector-space model, which we use in this paper. In our paper the vector $\vec{w_i}$ is associated with a word w_i as following: let us consider that $\{v_1, v_2, \cdots, v_m\}$ are m words of a high frequency in corpus. They can be of any POS set including prepositions and conjunctions from the closed class of words. The reason for this choice it is the known Zipf's result that a small set consisting of most frequent words can be used as framework for study of a natural language.

We define:

$$w_i^j = number\, of\, occurences\, of\, the\, word\, v_j\, in\, the\, same\, context\, with\, w_i$$
$$(for\, a\, number\, of\, contexts).$$

Received by the editors: February 7, 2003.

2000 *Mathematics Subject Classification.* 68T50, 68Q32.

1998 *CR Categories and Descriptors.* I.2.7 [**Computing Methodologies**]: Artificial Intelligence – *Natural Learning Processing*; G.3 [**Mathematics of Computing**]: Statistical Computing.

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

In [5], the vector $\vec{w_i} = (w_i^1, w_i^2, \cdots, w_i^m)$(where $m = 2 \times z$) associated with a word w_i, is constructed as follows: $w_i^j =$ number of occurrences of a word in the position $j = 1\,to\,z$ at left or $z + 1\,to\,m$ at right.

The paper is arranged as follows. Section 2 presents known clustering algorithms [6]: agglomerative algorithm for hierarchical clustering and divisive non-hierarchical k-means algorithm, adopted for our vector-space model. Section 3 proposes our variant for an agglomerative algorithm for hierarchical clustering such that a single "best" word is clusterized at a step. Section 4 describes how a list of similar words with a given word can be constructed. In section 5 we propose an experiment for Romanian language and present comparatively the results obtained by applying the clustering algorithms described in Section 2.

2. Clustering algorithms

Let us consider that the objects to be clusterized are the vectors of n words, $\{w_1, w_2, \cdots, w_n\}$. A vector

$$\vec{w_i} = (w_i^1, w_i^2, \cdots, w_i^m)$$

is associated with a word w_i as above.

Let us observe that the corpus must not be POS tagged or parsed since we are interested only of words and not of their syntactic role. However, we used a stammer to recognize the flexional occurrences of the same word (Romanian language is a very inflexional language).

The similarity measure between two words w_a, w_b is the *normalised cosine* between the vectors $\vec{w_a}$ and $\vec{w_b}$ [4]:

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

Agglomerative algorithm for hierarchical clustering [6]

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

Output The set of hierarchical clusters
$C = \{C_1, C_2, \cdots, C_n, C_{n+1}, \cdots, C_{n+k}\}$

begin

FOR $i = 1$ *TO* n *DO* $C_i = \{w_i\}$
$C = \{C_1, C_2, \cdots, C_k\}$
$j := n + 1$
$WHILE \mid C \mid > 1$ *DO*
$(C_{u^*}, C_{v^*}) := argmax_{(C_u, C_v)} sim(C_u, C_v)$
$C_j = C_{u^*} \cup C_{v^*}$
$C = C \setminus \{C_{u^*}, C_{v^*}\} \cup \{C_j\}$
$j := j + 1$

end

As similarity $sim(C_u, C_v)$ we considered:

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

The clustering algorithm begins by considering each word in its own cluster and ends when all the words are in the same cluster $C_{all} = C_{n+k}$. Let us consider $\{s_{n+1}, s_{n+2}, \cdots, s_{n+k}\}$ the values of similarities such that $s_i = sim(C_{u^*}, C_{v^*})$ and (C_{u^*}, C_{v^*}) has the same sense as in above algorithm. In other words, $\{s_{n+1}, s_{n+2}, \cdots, s_{n+k}\}$ are the values of similarities such that a new cluster $C_j = C_{u^*} \cup C_{v^*}$ is formed, $j = n + 1$ to $n + k$. The similarities $\{s_1, s_2, \cdots, s_n\}$ are all set to 1.

The similarities $\{s_1, s_2, \cdots, s_{n+k}\}$ are ordered decreasing from 1 (the similarities in clusters $C_i = \{w_i\}$, $i = 1, \cdots n$) to s_{n+k}, the similarity in the cluster $C_{all} = C_{n+k}$, as they occur on the dendrogram.

Non-hierarchical clustering algorithm: k-means algorithm [6]

Input The set $X = \{\vec{w_1}, \vec{w_2}, \cdots, \vec{w_n}\}$ of n vector words to be clusterised, the distance measure $d : R^m \times R^m \to R$, a function for computing the mean $\mu : P \to R$, the coefficient σ.

Output The set of clusters $C = \{C_1, C_2, \cdots, C_k\}$

begin

Select k initial centroids $\{\vec{f_1}, \vec{f_2}, \cdots, \vec{f_k}\}$
WHILE the diameter of a cluster $\geq \sigma$ *DO*
FOR all clusters C_j *DO*
$C_j = \{\vec{x_i} \mid \forall \vec{f_l}\ d(\vec{x_i}, \vec{f_j}) \leq d(\vec{x_i}, \vec{f_l})\}$
FOR all clusters C_j *DO*

$\vec{f_j} = \vec{\mu}(C_j)$

end

As distance measure we considered:

$$d(\vec{w_a}, \vec{w_b}) = \frac{1}{sim(\vec{w_a}, \vec{w_b})}$$

and as centroid:

$$\vec{\mu}(C_j) = \frac{1}{\mid C_j \mid} \sum_{\vec{x} \in C_j} \vec{x}$$

A diameter of a cluster we define as *the distance between the least similar elements in a cluster.*

3. An incremental algorithm for clustering

The following algorithm has the property that at the begin of the process it arrange at a time only one word to an appropriate cluster.

For a word w_i let $N(w_i)$ be the set of words w_j such that $sim(w_i, w_j) \neq 0$. For a set of words C, $N(C)$ will denote $\bigcup_{w_i \in C} N(w_i)$. The set $N(C)$ is similar with the set of *neighbours* of C in [9] but there the problem is solved on a graph model.

Let C be a set of words from W and $u \in N(C) \backslash C$.

We define

$$sim(u, C) = \sum_{w \in C} sim(w, u).$$

The best node u' from a set Q of words, which can be added to the set C, denoted $Best(C, Q)$, is the node which maximizes $sim(u, C)$:

$$Best(C, Q) = argmax_{u \in Q \cap N(C) \backslash C} sim(u, C).$$

Our hierarchical algorithm differs of the above hierarchical algorithm by the fact that in a step we form a new cluster by adding to a cluster C of the only word $Best(C, Q)$.

The algorithm is:

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

Output The set of hierarchical clusters
$C = \{C_1, C_2, \cdots, C_n, C_{n+1}, \cdots, C_{n+k}\}$

begin

$FOR\ i = 1\ TO\ n\ DO\ C_i = \{w_i\}$
$C = \{C_1, C_2, \cdots, C_n\}$
$Q = X$
$j := n + 1$
$WHILE\ Q \neq \Phi DO$
$s = argmax_{k=1,\cdots,j-1} Best(C_k, Q)$
$u' = Best(C_s, Q)$
$C_j = \{u'\} \cup C_s$
$C = (C \setminus \{C_s\}) \cup \{C_j\}$
$Q = Q \setminus \{u'\}$
$j := j + 1$

end

Let us remark that after all words from X (Q initial) are clustered, the algorithm stops.

Let us mention that our algorithm consider only a sense of a word u and for it exists only a cluster C such that $u = Best(C, Q)$. Of course this is not the case for polysemous words. In [9] is established that if G is a graph of words build on the base of a symmetric syntactic relation, and $G \setminus w$ is the subgraph which results from the removal of w, then the connected components of the subgraph $G \setminus w$ correspond to the senses of the word w. The above algorithm can be adopted in this sense, the symmetric relation being *sim*.

Once that some measure of similarity between words are established, we can begin a new process of divisive splicing in clusters. We seek to partition the set W of words into two subsets W_1, W_2 of the same size so that the similarity between W_1, W_2 is minimal: that means that

$$(W_1, W_2) = argmin_{V_1, V_2} \sum_{w_i \in V_1} \sum_{w_j \in V_2} sim(w_1, w_2)$$

An algorithm for implement a such of partition is a variant of hill-climbing search ([3]): after guessing an initial partition (W_1, W_2) we exchange two words between W_1 and W_2 if the exchange minimize $\sum_{w_i \in V_1} \sum_{w_j \in V_2} sim(w_1, w_2)$. We stop when no further decrease is possible.

4. The list of similar words for a given word

Input The set of hierarchical clusters $C = \{C_1, C_2, \cdots, C_n,\ C_{n+1}, \cdots, , C_{n+k}\}$ (as above), the set of similarities $\{s_1, s_2, \cdots, s_{n+k}\}$, a word $w \in X$

Output The lists *Elem* and *SimDecr* containing the elements in X in decreasing order of similarity with w and the sequence of these similarities.

begin
Set $j = 1$, $Elem(1) = w$ and $SimDecr(1) = 1$
FOR i=n+1 TO n+k DO
IF $w \in C_i$ *(* $C_i = \{C_{i,1}, \cdots, C_{i,p_i}\}$*) THEN*
FOR t=1 TO p_i *DO*
IF not$(C_{i,t} \in Elem)$ *THEN*
$j := j + 1$
$Elem(j) = C_{i,t}$; $SimDecr(j) = s_i$
end

A corresponding algorithm for calculating the list of similar words for a given word can be imagined using the k-means algorithm : for each word w, the words in the same cluster (let say C), in order of distances to w, begin the list. That list contains then the words from the others clusters, in order of distance (the inverse of similarity) from C. The similarity is: $sim(C_u, C_v) = \frac{\sum_{a_i \in C_u} \sum_{b_j \in C_v} sim(a_i, b_j)}{|C_u| \times |C_v|}$.

5. Results and evaluation

5.1. **Applications.** In this section we want to show how the clustering process (based on the algorithms described in the previous section) works.

The first application uses the non-hierarchical clustering algorithm (NHCA - section 2), the second uses the hierarchical clustering algorithm (HCA - section 2).

Both NHCA and HCA are written in JDK 1.4. The aim is to clusterize a set of words.

The NHCA algorithm starts with a set of contexts, a set of words having a maximum frequency in the given contexts and with a set of "focus" words (*terms*) used in the clustering process. As a result of the clustering, the algorithm reports a set of clusters (in a cluster will be the *similar* words - the words having similar senses).

The process starts with a set of initial clusters (based on the *focus* words), and after that, learns, based on the information obtained from the initial contexts to clusterize the set of words.

We have to notice that the set of *terms* used for the clustering is very important (this was shown experimentally).

The HCA algorithm starts with the same initial information as NHCA, except the set of *terms*. As a result of the clustering, the algorithm reports, for each word

w, a cluster that will contain the *similar* words with w, in descending order after their similarities.

Because in the HCA algorithm the process does not depend on a set of *focus* words, the clustering result is more exact than the result of NHCA algorithm.

It is obvious, for both algorithms, that if the number of contexts grow, the clustering's precision grows, too (this is shown experimentally).

The initial information, for both algorithms, is read from a text file.

5.2. **The applications design.** The basis classes used for implementing the two applications are the same; differs only the clustering algorithm. The main classes are:

- **CList**: defines the type the structure of a list of objects, having methods for:
  - adding an object in the list;
  - accessing elements from the list;
  - updating elements from the list;
  - returning the dimension of the list;
- **CLine**: defines the structure of a list having as elements real values (is defined using the *CList* class);
- **CContext**: defines the structure of a list having as elements words (is defined using the *CList* class);
- **CLine**: defines the structure of a list having as elements lists with real values (is defined using the *CList* class);

5.3. **Experimental results.** In this section we propose an experiment for the Romanian language: the aim is to clusterize a set of words (to group the words after the similarity of their meanings). We have applied both the NHCA and the HCA algorithms.

We mention that we used a set of 26 contexts. We also note that if we grow the number of contexts, the clustering's precision grow.

The initial information (the set of words to be clusterized, the contexts, the focus words) is read from a text file having the following structure:

---

the words to be clusterized

oameni oras durata timp partid persoana localitate perioada organizatie sat asociatie

a set of words that the clustering process is based on (at us, these are words having a maximum frequency in the contexts)

de in la sa care ca pe munca premier

the contexts

(1) indreptatirea la masurile reparatorii prevazute de prezentul articol este conditionata de continuarea activitatii ca persoana juridica pana la intrarea in vigoare a prezentei legi sau de imprejurarea ca activitatea lor
(2) In vederea desfasurarii anchetei disciplinare, salariatul va fi convocat in scris de persoana imputernicita de catre conducatorul unitatii sa realizeze ancheta
(3) Memoriul a ajuns la scoala din localitate, la primarie, la prefectura si la Insepectoratul Scolar al judetului Harghita
(4) Totodata, la Conel, persoanele detasate la unitati din alta localitate, precum si cele delegate in afara locului de munca au castiguri uriase
(5) ...

the focus words
persoana localitate perioada organizatie

After applying the NHCA algorithm, we obtained the following clusters:

Cluster 1 *timp partid persoana sat*
Cluster 2 *oras localitate*
Cluster 3 *durata perioada*
Cluster 4 *oameni organizatie asociatie*

As a measure for evaluation of the NHCA algorithm we propose the *precision* of the clustering, defined as follows:

$$P = \frac{\sum_{i=1}^{k} \frac{n_i}{N_i}}{k} \tag{1}$$

where k is the number of clusters, n_i is the number of words correctly placed in the i-th cluster, and N_i is the total number of words placed in the i-th cluster.

We mention that for our experiment, the precision of the NHCA algorithm is 93%.

For the same set of words to clusterize, we have applied the HCA algorithm.

The result obtained for the word *asociatie* is given in Table 1 (each word is followed by its similarity with the given word)

We mention that we ran the clustering algorithms on bigger data sets (10000 contexts), 200 words to clusterize an the results are very good.

We also mention that this clustering applications are part of a QA system that is developed for the Romanian language [7].

Word	Similarity
asociatie	1.0
oameni	0.8498365855987975
oras	0.6255587777150006
localitate	0.6255587777150006
organizatie	0.6255587777150006
timp	0.6255587777150006
persoana	0.6255587777150006
sat	0.6255587777150006
durata	0.5183688447475575
perioada	0.5183688447475575
partid	0.31611039139928965

TABLE 1. The result of applying the HCA algorithm for the word *asociatie*

6. Conclusion and future directions

The above algorithms must be connected with the word sense disambiguation algorithms [10] to work well with ambiguity. A WSD algorithm must be run to distinguish between two (or more) different senses of a polysemic word. In this case, the different occurrences of senses correspond to different words.

We intend to evaluate the QA system [7] by expanding of query terms, word by word, with most similar words in the lists.

We intend to use the similarity between two words to disambiguating a group of two words: it is well known that two polysemic words are better disambiguated when they occur together (for example *doctor* and *nurse* which are both polysemic [8]).

References

[1] S. A. Caraballo, "Automatic construction of hypernym-labeled noun hierarchy from text", Proceedings of ACL, 1999.
[2] I. Dagan, L. Lee, F. C. N. Pereira, "Similarity-based models of Word Coocurences Probabilities", MLJ 34 (1–3), 1999.
[3] Y. Even-Zohar, D. Roth, D. Zelenko, "Word prediction and Clustering", http://citesser.nj.nec.com/even-zohar99word.html
[4] D. Jurafsky, J. Martin, "Speech and language processing", Prentice Hall, 2000.
[5] J. Karlgren, M. Sahlgren, "From words to understanding", CSLI 2001, pp. 294–308.
[6] C. Manning, H. Schutze, "Foundation of statistical natural language processing", MIT, 1999.
[7] C. Orăşan, D. Tătar, G. Şerban, D. Avram, A. Oneţ, "How to build a QA system in your back-garden: application to Romanian", EACL '03, Budapest, April 2003.

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

[9] D. Widdows, B. Dorow, "A graph model for unsupervised lexical acquisition".

[10] G. Şerban, D. Tătar, "Word Sense Disambiguation for Untagged Corpus: Application to Romanian Language", Proceedings of CICLing 2003 (Intelligent Text Processing and Computational Linguistics), Mexico City, Mexic, Lecture Notes in Computer Science N 2588, Springer-Verlag, 2003, pp.270-275.

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

E-mail address: dtatar@cs.ubbcluj.ro, gabis@cs.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLVIII**, Number 1, 2003

APPLICATIONS OF SPATIAL DATABASES AND STRUCTURES TO THE STUDY OF MIOCENE DEPOSITS OF BOROD BASIN

MIRELA POPA AND MARIA GABRIELA TRÎMBIŢAŞ

Abstract. In this paper we apply spatial database and structure to render the paleo-relief of the Borod Basin. The boreholes data are stored in a spatial database. For the effective surface rendering we use local-Shepard interpolation with variable radius, based on a spatial grid and Delaunay triangulation. The generated pictures are more realistic compared to picture generated by mean of other methods.

1. Local Shepard Interpolation

The classical Shepard operator (see [19]) defined by

$$(S_{n,\mu}f)(x) = \sum_{k=0}^{n} w_k(x) f(x_k) \tag{1}$$

$$w_k(x) = \frac{|x - x_k|^{-\mu}}{\sum_{k=0}^{n} |x - x_k|^{-\mu}}, \tag{2}$$

where $|.|$ denotes the Euclidean norm in $\mathbb{R}^s$, and $X = \{x_0, x_1, \ldots, x_n\} \subset \mathbb{R}^s$ is a set of $n+1$ pairwise distinct points, requires a large amount of computation. The volume of computation can be reduced replacing the weight functions given by (2) with the so called Franke-Little weights:

$$\bar{w}_k(x) = \frac{\dfrac{(R - |x - x_k|)_+^{\mu}}{R^{\mu}|x - x_k|^{\mu}}}{\displaystyle\sum_{i=0}^{n} \frac{(R - |x - x_i|)_+^{\mu}}{R^{\mu}|x - x_i|^{\mu}}} \tag{3}$$

Received by the editors: February 10, 2003.

2000 *Mathematics Subject Classification.* 65D05, 65D18.

1998 *CR Categories and Descriptors.* G.1.1 [**Mathematics of Computing**]: Numerical Analysis – *Interpolation formulas*; H.2.8 [**Information Systems**]: Database Management – *Spatial databases and GIS*.

(see [14, 12, 13]). In (3), R is a given positive real constant, and the + subscript denotes the positive part. Thus we obtain the *local Shepard-operator*:

$$\left(\bar{S}_{n,\mu}^{L} f\right)(x) = \sum_{k=0}^{n} \bar{w}_k(x) f(x_k). \tag{4}$$

This operator reproduces the values of f in x_k and has the degree of exactness equal to zero, that is reproduces the constant.

In order to increase the degree of exactness one tries to replace the values $f(x_k)$ with the values of an interpolation operator: Taylor [9, 11, 8, 3, 10, 6], Lagrange [4, 5, 6], Hermite [1, 5, 6], Birkhoff [2, 5, 6], least square approximation [18, 16, 17, 21] and even spline[6]. The operators obtained in this way are called *combined Shepard operators*.

In this paper we are interested in simple local Shepard operator, given by (4).

2. Spatial data structures

In order to compute the various local Shepard-type interpolants we are interested to report efficiently the point located into the ball $B(x, R)$. The naive approach (computing $d_k = |x - x_k|$ and checking $d_k < R$) needs a time $O(n)$ for each point x. Computational geometry techniques and data structures allow us to perform this task in polylogarithmic time.

Let $P := \{p_1, \ldots, p_n\}$ be a set of point from $\mathbb{R}^s$ and Reg a region from the same space. A s-dimensional *range searching problem* asks for the points from P lying inside the query region Reg. If the region is a hyperparallelopiped, i.e. $Reg = [x_1, x_1'] \times \cdots \times [x_s, x_s']$, then we have an *orthogonal range-searching problem*. If Reg is a ball from $\mathbb{R}^s$, we have a *circular range searching problem*. Our approach is to solve a simpler orthogonal range searching problem instead the circular range searching (since this approach eliminates a large number of points) and then to check the reported points.

One of the most used data structure for orthogonal range query is the *range tree*[7]. A solution based on range tree is given in [20].

Another solution is inspired from a paper of Renka[18] and presented extensively in [21]. The smallest bounding box containing the interpolation nodes $\prod_{k=1}^{s} \left[x_{\min}^k, x_{\max}^k\right]$ is partitioned into an uniform grid of cells, having NR cells on each dimension. Each cell points to the list of point indices contained in that cell. Such an example for the 2D case is given in Figure 1. The algorithm 1 describes the creation of the data structure. If the second argument NR is not provided, we can initialize it with a default value; Renka suggests in [16]

$$NR = \lfloor (N/3)^{1/\dim} \rfloor.$$

The orthogonal range searching is easy to implement using this data structure (the algorithm 2): first the cell which must be scanned are determined (i.e. the cell

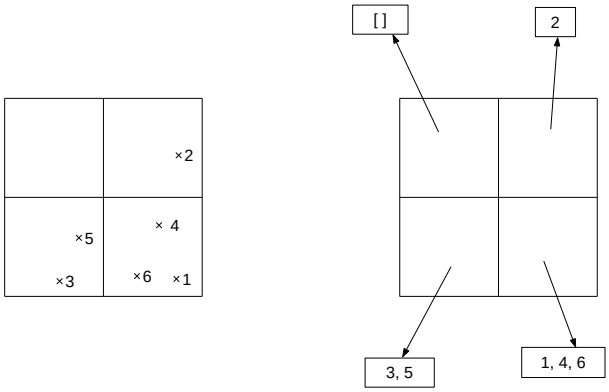


FIGURE 1. A 2D grid of cell and its representation

which intersects the searching domain), and then the list of points corresponding to that cell are concatenated. The points from the outer cells which lie outside the searching range must be eliminated.

Algorithm 1 Creating the cell grid

Input: the set of N points P, the number of cells, NR (optional);
Output: a grid of cell LCELL, each containing the list of points in the cell
set all cells to **nil**;
{compute the cell sizes}
$dc_1 := \min(NR, \lfloor x^1_{\max} - x^1_{\min} \rfloor + 1)$;
$\vdots$
$dc_s := \min(NR, \lfloor x^s_{\max} - x^s_{\min} \rfloor + 1)$;
for $K := N$ **downto** 1 **do**
  {find the cell}
  $i_1 := \min(NR, \lfloor x^k_1 - x^1_{\min} \rfloor + 1)$;
  $\vdots$
  $i_s := \min(NR, \lfloor x^k_s - x^s_{\min} \rfloor + 1)$;
  add K to the list $LCELL(i_1, \ldots, i_s)$;
end for

Now we are able to compute the local Shepard interpolant on a set of points X:

- build the spatial data structure;
- for each point x in X

– perform the orthogonal range searching into the hypercube centered in x and with the radius R
– apply formulas (3) and (4).

Algorithm 2 The orthogonal range searching

$PTLIST :=$ **nil**;
{determine the outer cells, i. e. the scan limits}
$imin_1 := \max(1, \lfloor (liminf_1 - x^1_{\min})/dc_1 \rfloor + 1)$;
$imax_1 := \min(NR, \lfloor (limsup_1 - x^1_{\min})/dc_1 \rfloor + 1)$
$\vdots$
$imin_s := \max(1, \lfloor (liminf_s - x^s_{\min})/dc_s \rfloor + 1)$;
$imax_s := \min(NR, \lfloor (limsup_s - x^s_{\min})/dc_s \rfloor + 1)$
for $i_1 := imin_1$ **to** $imax_1$ **do**
  $\vdots$
  for $i_s := imin_s$ **to** $imax_s$ **do**
    $JL := LCELL(i_1, \ldots, i_s)$;
    if the cell $(i_1, \ldots, i_s)$ is peripheral **then**
      remove the points which lay cell outside the searching range from JL;
    end if
    concatenate $PTLIST$ and JL
  end for
  $\vdots$
end for

This approach has a drawback: the accuracy tends to decrease into the areas where the interpolation nodes are sparse. We can avoid this situation, allowing the radius R to vary with k: the radii are chosen such that the ball $B(x, R)$ contains at least N_w nodes. Thus, instead of an orthogonal range searching we perform a N_w-th nearest neighbor search of x_j and x, respectively. This can be done scanning the grid in a circular fashion starting with the cell containing x. In order to facilitate the scanning we can associate a Boolean indicator to each cell, which is true when the cell was already scanned.

3. The geological data

Borod Depression is located in the western part of the Apuseni Mountains, being bordered by Plopisului Mountains in the north and by Padurea Craiului in the south.

This depressionary area was formed about 12-16 million years ago, during the Badenian, along a fault located on the northern border, in contact with the Plopis

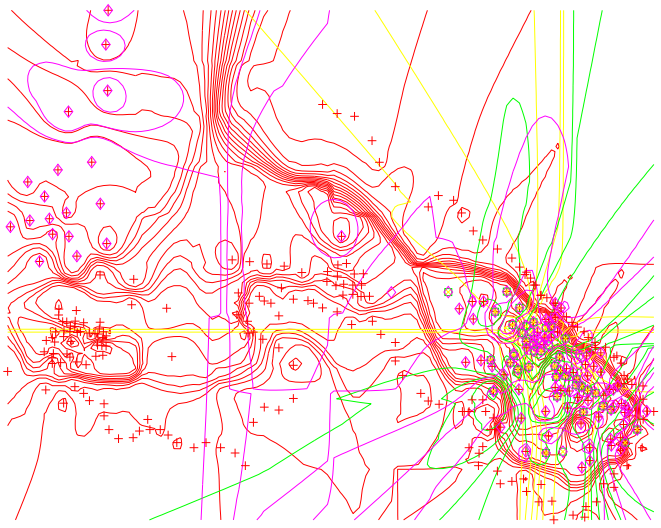


FIGURE 2. Geological boreholes

Mountains. Subsequently it represented a sedimentary basin where a thick succession of sediments (200-1000m) was accumulated.

The Neogene deposits representing the filling of the basin were assigned to three different formations: Borod Formation (Badenian), Cornitel Formation (Sarmatian), and Beznea Formation (Pannonian) [15]. Each formation (unit) consists of banks of rocks with a variable thickness, separated according to well-established criteria.

This area was investigated during the last three decades by using geological drilling (Figure 2. Geological studies based on borehole data imply the drawing of:

- longitudinal and transversal geological profiles
- Isogram maps (isobath and isopachyte)
- 3-D modelling of the paleorelief at various levels (basement and top of the formations);

For each borehole the database contains the following information:

- The borehole ID;
- The abscissa and the ordinate of the borehole;
- From one to four z-coordinates representing the borehole depth, corresponding to Basement, Badenian, Sarmatian, and Pannonian age.

The graphical representations offer a suggestive image on the specific features of the basin formation and on its evolution in time. In the previous decades the graphs – except for the 3-D – were performed manually. This stage was partly overcome due to attempts to electronic processing of data, including 3-D modelling. It is worth to mention that on an international level, computer graphics is currently a common tool in geological sciences.

Our previous trials to draw 3-D block diagrams by using other types of operators were not satisfactory. On the contrary, the method presented in the paper clearly evidences the fractured areas (faults), the space arrangement of the geological blocks, and the relationships among the various formations. The modelling of the paleoenvironment at the basement level evidences the fault along the northern border that shaped the basin formation. In the same time, the sets of faults that shaped the pre-Neogene deposits (older than 65 million years) are also noticeable. The graphs obtained for the top of the formations (Borod-yellow, Cornitel-green, and Beznea-magenta) suggest the presence of some faults that have influenced also the geological structure of these deposits. Some of these faults represent older, basement-related ones that were subsequently reactivated; others are younger and were probably generated by petrographical discontinuities in the deposits that form the cover (Figures 3, 4).

A "scaled" reconstruction of a sedimentary basin at different stages of its evolution is an extremely useful tool in the geological research. 3-D block diagrams represent the most suggestive image of a basin at various formation stages if the method of interpolation of data between different drill locations used was based on the suitable operators.

The variable radius (range) local Shepard operator based on a rectangular network (grids) as well as the Delaunay triangulation definitely provide a more suggestive image on the spatial relationship between geological blocks.

The graph already presented are build on a rectangular grid. A more useful and realistic manner is to generate surfaces over a polygonal convex area. This can be achieved using a triangular grid the Delaunay triangulation (see [7]). The idea is to consider a polygonal area and a sufficiently large number of points in this area; we compute the function values on these points and then the Delaunay triangulation for this set of points. The membership of a point to a region is easily performed with this data structure. Finally, we render the surface on the triangular grid computed in this way.

Figures 5 and 6 give such representations for the surfaces given in Figures 3 and 4, respectively.

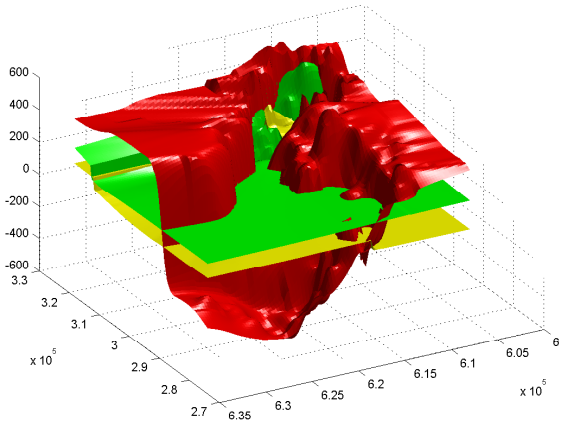


FIGURE 3. Representation: Basement – red, Badenian (top of Borod Formation) – yellow, Sarmatian (top of Corniţel Formation) – green

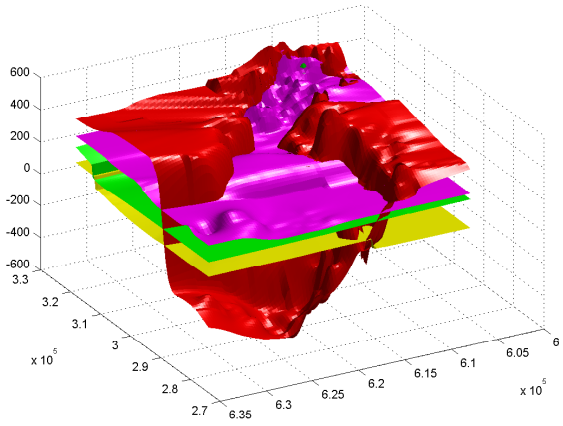


FIGURE 4. Representation: Basement – red, Badenian (top of Borod Formation)– yellow, Sarmatian (top of Corniţel Formation) – green, Pannonian (top of Beznea Formation) – magenta

4. CONCLUSIONS

The variable radius Shepard interpolation is a feasible approach for scattered data interpolation. Using spatial data structures leads us to efficient algorithms

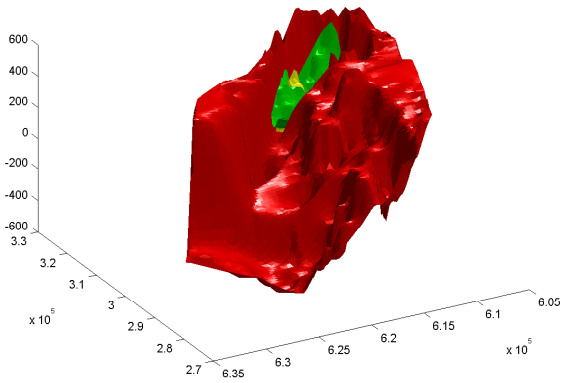


FIGURE 5. Representation of the surfaces from Figure 3 using Delaunay triangulation

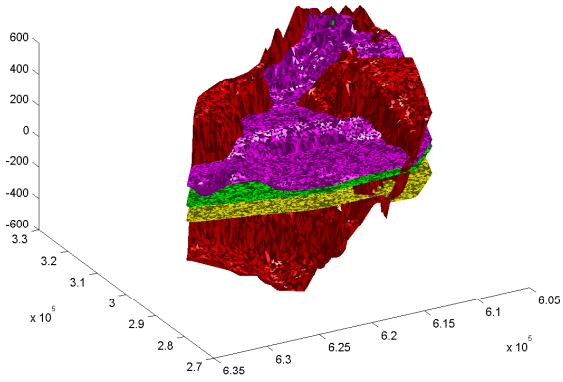


FIGURE 6. Representation of the surfaces from Figure 4 using Delaunay triangulation

for computing such interpolation operators and rendering the corresponding operators.

The method is also suitable from application point of view. It provides a more suggestive image on the spatial relationship between geological blocks and reveals some faults which other previous trials cannot reveal.

References

[1] Gh. Coman. Shepard operators of Hermite-type. *Rev. Anal. Numér. Théor. Approx*, 26:103–111, 1997.

[2] Gh. Coman. Shepard operators of Birkhoff-type. *Calcolo*, 35(4):197–204, 1998.

[3] Gh. Coman and L. Ţâmbulea. A Shepard-Taylor approximation formula. *Studia Univ. "Babeş-Bolyai"*, XXXIII(3):65–73, 1988.

[4] Gh. Coman and R. Trîmbiţaş. Shepard operators of Lagrange-type. *Studia Univ. "Babeş-Bolyai", Mathematica*, XLII(1):75–83, 1997.

[5] Gh. Coman and R. Trîmbiţaş. Bivariate Shepard interpolation. *"Babeş-Bolyai" University, Faculty of Mathematics and Computer Science, Seminar on Numerical and Statistical Calculus, Preprint*, (1):41–83, 1999.

[6] Gh. Coman and R. Trîmbiţaş. Combined Shepard interpolation. *East Journal of Approximation Theory*, 7(4):471–483, 2001.

[7] Mark de Berg, Mark van Kreveld, Mark Overmars, and Otfried Schwarzkopf. *Computational Geometry, Algorithms and Applications*. Springer Verlag, Berlin, Heidelberg, New York, Tokio, 1997.

[8] B. Della Vecchia and G. Mastroianni. Pointwise estimates of rational operators based on general distribution of knots. *Facta Universitatis, Ser. Math. Inform.*, 6:63–72, 1991.

[9] B. Della Vecchia and G. Mastroianni. Pointwise Simultaneous Approximation by Rational Operators. *Journal of Approximation Theory*, 65:140–150, 1991.

[10] B. Della Vecchia and G. Mastroianni. On functions approximations by Shepard-Type operators - a survey. In *Approximation theory, wavelets and applications, Maratea, 1994*, volume 454 of *Ser. C, Math. Phys. Sci.*, pages 335–346, Dordrecht, 1995. NATO Adv. Sci. Inst., Kluwer Acad. Publ.

[11] R. Farwig. Rate of Convergence of Shepard's Global Interpolation Formula. *Math. Comp*, 46(174):577–590, 1986.

[12] R. Franke. Scattered data interpolation: Tests of some methods. *Math. Comp.*, 38:181–200, 1982.

[13] R. Franke and G. M. Nielson. Smooth interpolation of large sets of scattered date. *International Journal for Numerical Methods in Engineering*, 15:1691–1704, 1980.

[14] R. Franke and G. M. Nielson. Scattered Data Interpolation and Applications: A Tutorial and Survey. In D. Roller H. Hagen, editor, *Geometric Modeling - Methods and Applications*, Berlin, Heidelberg, New York, 1990. Springer Verlag.

[15] Mirela Popa. Lithostratigraphy of the Miocene Deposits in the Eastern Part of Borod Basin (North-Western of Romania). *Studia Universitatis "Babeş-Bolyai", Geologia*, XLV(2):95–103, 2000.

[16] R. J. Renka. Algorithm 660, QSHEP2D: Quadratic Shepard Method for Bivariate Interpolation of Scattered Data. *ACM Transactions on Mathematical Software*, 14(2):149, 1988.

[17] R. J. Renka. Algorithm 661, QSHEP3D: Quadratic Shepard Method for Trivariate Interpolation of Scattered Data. *ACM Transactions on Mathematical Software*, 14(2):151, 1988.

[18] R. J. Renka. Multivariate Interpolation of Large Sets of Scattered Data. *ACM Transactions on Mathematical Software*, 14(2):139–148, 1988.

[19] D. Shepard. A Two Dimensional Interpolation Function for Irregularly Spaced Data. In *Proc. 23rd Nat. Conf. ACM*, pages 517–523, 1968.

[20] **M. G. Trîmbiţaş** and R. Trîmbiţaş. Range searching and local Shepard interpolation. In Al. Lupaş, editor, *Proceedings of Romanian-German Seminary (ROGER) 2002*, pages 279–292, Sibiu, 2002.

[21] M. G. Trîmbiţaş. Combined Shepard-least square operators – Computing them using spatial data structures. *Studia Univ. "Babeş-Bolyai", Mathematica*, XLVII(4):119–128, 2002.

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

Faculty of Mathematics and Computer Science, Babeş-Bolyai University, 1, Kogălniceanu St., RO-3400 Cluj-Napoca, Romania

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

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLVIII**, Number 1, 2003

AN APPROACH ON SEMANTIC QUERY OPTIMIZATION FOR DEDUCTIVE DATABASES

ADRIAN ONEŢ

Abstract. In this article we present a learning method to obtain rules for the semantic query optimization in deductive databases. Semantic query optimization can dramatically speed up deductive database query answering by knowledge intensive reformulation. We will present a learning method for rules that will help to semantically optimize queries for deductive databases.i We tried to change the algorithm in [2] to work for deductive database as well in this direction we propose a method for an approximate cost evaluation for deductive database predicates.

1. Introduction.

The semantic query optimization (SQO) is based on the use of the semantic rules, as it is *There is no river trough Carei city.* Using these rules, we can reformulate the queries in lower cost ones, for example: *Which are the cities that are in the Carei neighborhood?* (in the following we will consider that two cities are neighbors if there exists a river that connects these two cities). Using the semantic rules we can answer this query even without accessing the deductive database, so we will obtain a 100% cost reduction. Average savings from 20 to 40 percents are reported in the literature.

Unlike other systems, conceived for deriving rules from one database table, the Hsu&Block inductive method [2] of learning can learn semantic rules from a database with several relations (and, in our case, by using several base or derived predicates). For instance, if we consider a database that contains three relations *pupils, bachelor_grades, coordinator*, the bachelor grades can concord with the papers coordinator (e.g. All who have Mestereanu as coordinator obtained grades superior to 9 at the bachelor exam). The learning algorithm can select the relevant ways from the disjunction of two or more predicates (which is often made by the user). By using the semantic relations that describe some regularities in the disjunction of two or more predicates, our optimizer will be more than efficient

Received by the editors: March 15, 2003.

2000 *Mathematics Subject Classification.* 68U35.

1998 *CR Categories and Descriptors.* I.2.1 [**Computing Methodologies**]: Artificial Intelligence – *Appplications and Expert Systems*.

in the diminution of more complex queries execution price. We need to underline that this mechanism is profitable in a static database (without many updating in the extensional database as well as in the intensional one).

2. The semantic query optimization

The semantic query optimization is applied to different database types. Although the basic pattern of the semantic optimization refers only to the conjunctive queries, it can also be extended to other types, more complex. The general idea is that the complex queries can be decomposed in one or more conjunctive queries; after that the system can apply the semantic optimization of these queries. In this chapter we'll focus only on the conjunctive queries.

For an easier comprehension, we'll use the following knowledge database:
The extensional database structure:
Town [Name, Components]
Descriptions [Type_Component, Description]
Integration [Component, Type_Component]
Fraternity [Name_Town1, Name_Town2, Date]

with the following meaning: the relation **town** contains the name and the different components in this town (e.g. the town: *Bucharest* has the component: *Art_Museum*). The second relation, **descriptions**, obtains, for each component type a description of the respective town (e.g. if the component *Art_Museum* belongs to the component type *Historical_Objective* - according to the relation **integration** - and if in the relation description we have the tuple: Type_Component: *Historical_Objective* Description: *Tourism*, than we can say that Bucharest town is a touristic town). As we have already shown, the relation integration tells us to which type_component belongs a component by a transitivity relation. Finally, the relation **fraternity** gives us information about towns which are united, such as the date when they established the fraternity.

The intensional database structure: (in order to describe the intensional database, we maintain the Prolog syntax regarding to the variables and constants name)

objectives(Locality, Objective) :-
town(Locality, Component),
type_objective(Component, Type_component),
descriptions(Type_component, Objective).

type_objective(Component, Type_component) :-
integration(Component, Type_component).

type_objective(Component, Type_component) :-
integration(Component, Type_component1),
type_objective(Type_component1, Type_component).

We consider the following data in the extensional database:

Town

CAREI	CASTLE
CAREI	PARK
TURDA	FACTORY
CLUJ	SQUARE
CLUJ	BOTANIC_GARDEN
DEJ	FACTORY

Descriptions

HISTORICAL_OBJECTIVE	TOURISM
WORKS	POLLUTION
GREEN_SAPCES	BEAUTY
COMMERCIAL_PLACES	BUSINESS

Integration

CASTEL	MONUMENT
MONUMENT	HISTORICAL_OBJECTIVE
FACTORY	WORKS
PARK	GREEN_SPACES
SQUARE	COMMERCIAL_PLACES
BOTANIC_GARDEN	GREEN_SPACES

Fraternity

CAREI	DEJ
TURDA	CLUJ
CLUJ	DEJ

The main principle of the semantic query optimization is based on finding the equivalent queries for the initial query, but at a lower cost. The construction of the equivalent queries with the initial query is realized by using some semantic rules, which will be learned by the system from the previous queries. By a lower cost of the queries we understand a real cost approximation (the exact calculus of the cost would determine the diminution of the algorithm efficiency).

The difference between the syntactic optimization and the semantic one consists in using the semantic knowledge for expanding the search field for the semantic optimization. The conventional syntactic optimizations search the cheaper equivalent queries from the logical point of view for the initial queries [5] (the optimizations which re-sort literals/constraints belong to this category). The semantic optimization, on the other hand, searches the queries with the lowest cost equivalent to the initial query, by giving some semantic information. That is why, this type of optimization has a bigger search field and the lowest cost is also more probable comparing to the queries obtained by the syntactic optimization.

3. The learning model

In this sequence we try to present a learning model of the semantic rules (model presented in [2]). The figure 1 shows the database organization with a semantic optimizer and a learning system. The optimizer uses the semantic rules in order to optimize the queries and to send the optimized queries to rule on the deductive database in order to find the result. When the deductive database is dealing with a complex query (we mean expansive), the optimizer connects the learning system to learn a set of rules which are to be used for the optimization of other similar queries. The system will learn gradually sets of rules used for optimization.

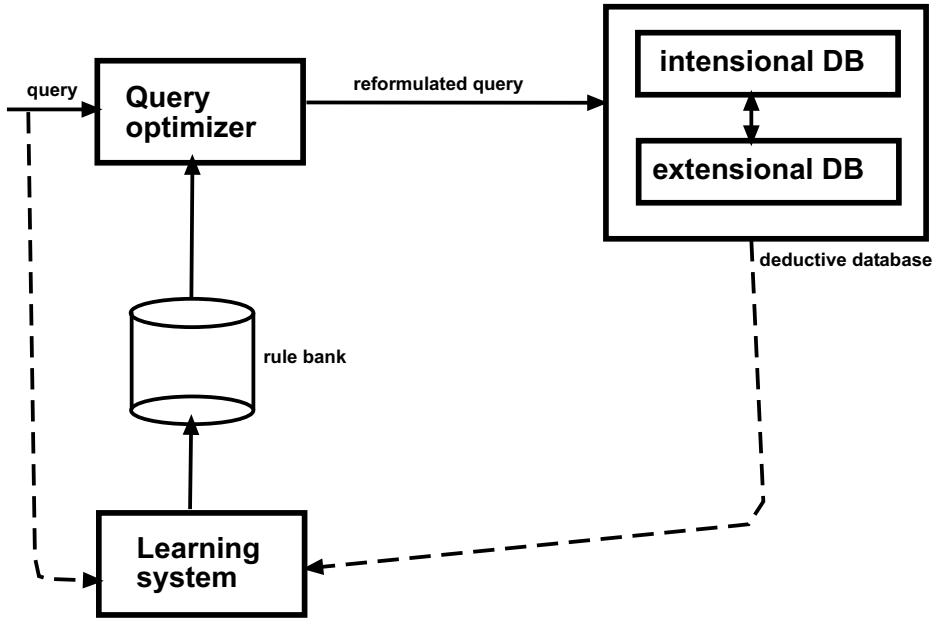


Fig.1 The structure of a knowledge database with a semantic optimizer

In the figure 2 there is a simple example of such a learning/optimization pattern. This model consists in two components [Chun95], a learning inductive component and an operational one. A query will call the learning component; afterward the system will apply an inductive learning algorithm to induct an alternative query of the initial query, but at a lower cost. Afterwards, the operationalization component, using the initial query and the alternative query learned, deduce a set of semantic rules.

In this example, the tuples from the relation are considered positive or negative depending of the query satisfaction or non satisfaction. The alternative query will

have to cover only the positive instances of the relation, thus the response to the alternative query will be the same as for the main query.

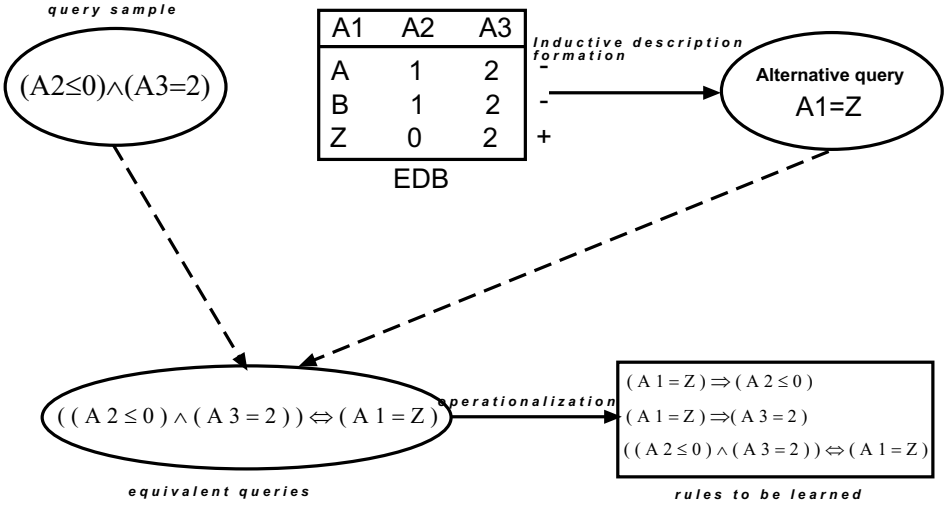


Fig.2 A learning pattern

Given a set of queries considered positive or negative, the problem of finding a description which covers only the positive instances, is named *supervised inductive learning* [2]. The more difficult problem is to compare the cost of different queries (when choosing the most appropriate semantic rule). If we use a calculus of the real cost, this optimization algorithm could not justify its objectives (the cost of such a calculus is big enough, sometimes even impossible to calculate). In this chapter we propose less expansive method which approximates the cost of such queries.

The operationalization component deduces semantic rules by using two equivalent queries. These two queries consist in two phases. In the first one, the system transforms the equivalence of the two queries into Horn clauses. For instance, given the queries: $(A2 \leq 0) \wedge (A3 = 2)$ and the equivalent query A1='Z', they will be transformed in two clauses:
1)$(A2 \leq 0) \wedge (A3 = 2) \Rightarrow A1 =' Z'$
2)$A1 =' Z' \Rightarrow (A2 \leq 0) \wedge (A3 = 2)$
The second rule can be expanded again in order to satisfy the Horn clause syntax:
3)$A1 =' Z' \Rightarrow (A2 < 0)$
4)$A1 =' Z' \Rightarrow (A3 = 2)$

After transformation, we obtain the rules 1), 3) and 4) which satisfy our syntactic demands. In the second phase, the system tries to compress the antecedents

of these rules for reducing their cost. Thus, if the rules have more than one antecedent, can be used the *greedy* minimal cover algorithm in order to eliminate the respective constraints. The minimal cover problem is to find a sub-set from a collection of sets, thus the reunion of the sub-set sets is equal to the reunion of all sets. Denying both of the first rule parts, we obtain:
$\neg(A1 =' Z') \Rightarrow \neg(A2 \leq 0) \vee \neg(A3 = 2)$

Thus, for the clause 1), we have the following problem: given a set collection which satisfy $\neg(A2 \leq 0) \vee \neg(A3 = 2)$ find the minimum number of sets which satisfy $\neg(A1 =' Z')$. If we suppose that the minimum set of the sets which covers $\neg(A1 =' Z')$ is $\neg(A2 \leq 0)$, then, in this case we can also eliminate $\neg(A3 = 2)$ from the rule and, after the clause denying, we obtain: $(A2 \leq 0) \Rightarrow A1 =' Z'$

4. Alternative query learning

In this sequence, we present an inductive learning method of the alternative queries with reduced cost. In the figure 1, it was given an example with only one predicate, but, usually, the deductive databases consist in more predicates (the basic databases as well as the derived ones), and the queries structure often implies relations of the join type. The inductive learning model described is able to learn conjunctive queries at reduced cost from the deductive databases with more predicates.

Before describing the model, we have to introduce two terms that describe the queries obtained, such as: **internal disjunctions** (constraints over one attribute value), **join constraints** (they specify a constraint over one or several attributes from different predicates) [4].

The learning model is an extension of the *greedy* algorithm which learns internal disjunctions from one database created with one table, algorithm proposed by Haussler [3]. This algorithm starts from an empty hypothesis of the concept to be learned, then continues by constructing a set of candidate constraints which respect all the positive instances, in order to choose the most promising constraints by using one heuristic function such *gain/cost*, which is added to the hypothesis. This process is repeated until negative instance doesn't satisfy the hypothesis.

The algorithm has as entrance the Q query and the predicates from the deductive database. We call the **base relation** the relation that must be accessed for the initial queries. If the output attributes of the query are connected to different predicates, then the base relation is the relation resulted from the join of these relations. For example, if, in our case, we consider the query *What towns have as main objective the beauty* the base relation would be given by the predicate **objectives**, the tuples of this relation (the ones marked by + are those who satisfy the query):

Objectives

CAREI	TOURISM	
CAREI	BEAUTY	+
TURDA	POLLUTION	
CLUJ	BUSINESS	
CLUJ	BEAUTY	+
DEJ	POLLUTION	

Initially, the system determines the base relation of the entrance query, then it marks its instances as being positive or negative (an instance will be marked as positive if it satisfies the query, and as negative on the contrary)

Algorithm Learning alternative queries
Input: Q- entrance query, DDB - deductive database
Output: AQ - alternative query
Let r = base relation for Q
Let AQ = $\oslash$ - alternative query
Let C = $\oslash$ - candidate constraints set
It builds candidate constraints for r and added to C
Repeat
Evaluates the rapport gain/cost of the constraints from C
Let c = the constraint with the lowest value for gain/cost from C
if gain(c)>0 **then**
add c to AQ
C = C - c
if AQ$\Leftrightarrow$Q **then return** AQ
else
if c is a constraint of the join type on the new relation r' **then**
build candidate constraints for r' and it is added to C
end if
end if
end if
until gain(c) = 0
return failure - because it wasn't found AQ equivalent to Q
end algorithm

In this algorithm we still have to explain how the constraints were built and how the calculus of the heuristic function *gain/cost* was made.

5. Building the candidate constraints

For every attribute from the base relation, the system can build an internal disjunction, as well as a candidate constraint, by generalizing the positive instances

attributes value. For the query given as example for the first attribute from the base relation we have the following values: *Carei*, *Cluj*, and, if we consider the second attribute, we have the value *beauty*. Similarly, the system can also consider the constraints of join type as well as the candidate constraints if it consists in all the positive instances. If, for example, we have join between **objectives** and **fraternity**, we'll obtain the relation (we marked with + the instances which satisfy the query)

Objectives⊗Fraternity

CAREI	TOURISM	CAREI	DEJ	
CAREI	BEAUTY	CAREI	DEJ	+
TURDA	POLLUTION	TURDA	CLUJ	
CLUJ	BUSINESS	CLUJ	DEJ	
CLUJ	BEAUTY	CLUJ	DEJ	+

We can notice that this kind of relation can help us finding a candidate constraint, for example for the second attribute of the predicate fraternity, we have, for all the positive instances the value DEJ (we have to consider, of course, the cost as well, but in most of the cases, the cost of one join is lower then the cost of traversal of one relation of the join [6]).

6. The evaluation of the candidate constraints

Once built the set of the candidate constraints, we will have to establish which is the most promising one and to add it to the hypothesis (see the algorithm). To do it, we'll have to evaluate the *gain/cost* value for each constraint, where by *gain* we mean the number of excluded negative instances. For the basic predicates there is one set of algorithms that approximate the cost function of their physical structure [5],[6]); in the case of derived predicates, their cost estimation is more difficult. We'll present as follows an estimation method for the cost of these predicates (the costs also depend, of course, on the basic predicates access).

The cost estimation method is a rewriting method. Consequently, we add a new basic predicate **estimated_cost(predicate, cost)** whose argument is the predicate, that is the estimated cost of the given predicate. The algorithm consists in the modification of the tuples corresponding to the relation function of the introduced relations (thus, as it is no longer necessary to access these predicates when calculating, but only to consult the relation corresponding to the predicate estimated_cost). Initially, for each derived predicate, will exist a corresponding tuple with the initial value is 1. For our example, we will have:

estimated_cost

objectives	1
type_objective	1

The next step consists in rewriting the rules so that they can evaluate the approximated cost of a predicate. To do this, we'll have to add another attribute to every derived predicate and that will represent the cost. Thus, this cost will increase at every call of the clause function of the dimensions of the relations corresponding to the other predicates and function of the dimension of the relations corresponding to the basic predicates (here we can optimize by introducing other parameters such as indexes [5]) which compose the respective clause. Thus, for our example, we'll have the following intensional database (after rewriting):

objectives(Locality, Objective, Cost_objectives) :-
town(Locality, Component),
type_objective(Component, Type_component, Cost_type_objective),
descriptions(Type_component, Objective).
Cost_objectives=dim_town * Cost_type_objective * dim_description.

type_objective(Component, Type_Component, Cost_type_objective) :-
integration(Component, Type_Component),
Cost_type_objective=dim_integration.

type_objective(Component, Type_Component, Cost_type_objective) :-
integration(Component, Type_Component1),
type_objective(Type_Component1, Type_Component,
Cost_type_objective_intermediary),
Cost_type_objective = dim_integration *Cost_type_objective_intermediary.

Where dim_town, dim_description, dim_integration are some variables representing the cost of the access to the relations: Objective, Description, and Integration. All that remains to do is to memorize the results in the relation corresponding to the predicate **estimated_cost**, which can be realized by using an update predicate (by updating one predicate value cost we'll understand the change with the cost biggest value) of the tuples at the call of each predicate. This is not recommended because there can exist internal predicates which are not called explicitly by the query, but called only by other predicates whose cost is not evaluated. That is why, we have chosen (for now - in the future, we are counting on finding a better solution) to integrate this update predicate in every clause, corresponding to the heading predicate. Thus, for the previous example, we have (supposing that the update predicate is named **cost_update(predicate, cost))**:

objectives(Locality, Objective, Cost_objectives) :-
town(Locality, Component),
type_objective(Component, Type_Component, Cost_type_objective),
description(Type_Component, Objective),
Cost_objectives=dim_town * Cost_type_objective * dim_description,
Cost_update(objectives, Cost_objectives).

type_objective(Component, Type_Component, Cost_type_objective) :-

integration(Component,Type_Component),
Cost_type_objective= dim_integration,
Cost_update(type_objective, Cost_type_objective).

type_objective(Component, Type_Component, Cost_type_objective) :-
integration(Component,Type_Component1),
type_objective(Type_Component1, Type_Component, Cost_type_objective_inter),
Cost_type_objective = dim_integration * cost_type_objective_inter,
Cost_update(type_objective, Cost_type_objective).

The predicate **Cost_update**/2, as we have specified, will change the predicate (the first parameter) cost value (the second parameter) in the relation corresponding to the predicate **estimated_cost**/2, only if the cost value given by the second parameter is bigger then the value found in the relation.

This algorithm works very good for the systems whose evaluation is bottom-up (it ignores the parameters link); in the top-down case, there are differences between the cost of a predicate which has no linked parameter and the cost of a predicate which has all the parameters linked. That is why, for the top-down evaluation systems, we change the previous algorithm so that this evaluation still makes the difference between the parameters different ways of linking. The modification consists in adding a new attribute to the relation **estimated_cost**/3 that tells us for what link type the cost is calculated. Consequently, in our example, the relation contains initially:

estimated_cost

objectives	bb	1
objectives	bf	1
objectives	fb	1
objectives	ff	1
type_objective	bb	1
type_objective	bf	1
type_objective	fb	1
type_objective	ff	1

We can notice that we only considered the parameters that entered in the initial relation. The clauses will be also modified, meaning that we will add one parameter from the list type to each derived predicate. This parameter will tell us about the way that the link is realised for the predicate, at that moment; thus, the way of a variable link also depends on the way that the previous predicate was evaluated. Our intensional base will be as follows:

objectives(Locality, Objective, Cost_objectives, Link_Objectives) :-
town(Locality, Component),
type_objective(Component, Type_Component, Cost_type_objective, [b,f]),
descriptions(Type_Component, Objective),

Cost_objectives=dim_town * Cost_type_objective * dim_descriptions,
Cost_update(objectives, Cost_objectives, Link_Objectives).

type_objective(Component, Type_Component, Cost_type_obj, Link_type_obj) :-
integrare(Component, Type_Component),
Cost_type_obj= dim_integrare,
Cost_update(type_objective, Cost_type_obj, Link_type_obj).

type_objective(Component, Type_Component, Cost_type_obj, Link_type_obj):-
integration(Component,Type_Component1),
value(2,Link_type_obj,L),
type_objective(Type_Component1,Type_Component,Cost_type_obj_inter,[b,L]),
Cost_type_obj = dim_integration*cost_type_obj_inter,
Cost_update(type_obj, Cost_type_obj,Link_type_obj).

We used the **value**/3 predicate that indicates the way of link of the parameter given by number by the first argument from the link list given as the second argument, and the third argument will make us return to the respective value. For example, the predicate **value**/3 description will be:
value(1,[A|_],A).
value(N,[_|T],A):- *succ*(N1,N), *value*(N1,T,A).

At every derived predicate call, we'll also have to precise the way of parameters linking (for instance, the query *What are the towns that have as principal objective beauty* is described as follows: ?:- objectives(Town, beauty, Cost, [f,b])).

We notice that, in this case, the **cost_update** predicate has three arguments (was also added the link list - a tuple from **estimated_cost** is once identified by the predicate name and by the way of the parameters linking).

In this algorithm constants were used for the cost of the relations corresponding to the main predicates, these constants are calculated according to the formula described by [6].

After this modification of the database, the cost calculus for the candidate constraints is no longer an issue, because we have already calculated these costs in the **estimated_cost** relation; thus, using one traversal of this relation, we can determine every derived predicate that belongs to a candidate constraint.

7. The search in the candidate constraints domain

When a join type constraint is chosen, a new relation is introduced in the candidate constraints domain. The system can choose to add new constraints on the account of the new relation from the hypothesis or to consider the attributes of the old relation. When a join type constraint is added, the domain is devised in two levels, if to the new introduced relation constraints a new join type constraint is added; then, the domain will be devised in three levels and so on. The exhaustive evaluation of the gain/cost for all the candidate constraints is not at all practical when we deal with a big and complex database. The system introduce by [2] prefers

the model that favors the candidate constraints of the new introduced relation. Thus, when a join type constraint is selected, the system will consider only those candidate constraints from the new level, until the system builds a hypothesis that covers all the positive instances and no negative instance (that means that it achieved its purpose), or until it can not find other constraints that have a positive cost on that level. In the last case, the system returns to the previous level, continuing searching. This evaluation method was chosen because a join type constraint is less probable chosen from the rest of the constraints (usually, the join constraints have a bigger cost); if this constraint was chosen, it means that the rest of the constraints have a very high cost or a negative gain.

8. Conclusion

In this article we have shown that the knowledge required for semantic query optimization can be learned under the guidance of the input queries. We have described a method to approximate the query execution cost in deductive databases. We can use this method in the inductive learning algorithm described by [2] for semantic query optimization in deductive databases. A limitation of this semantic query optimization approach is that there is no mechanism to deal with changes in the deductive database. This problem can be solved as follows: when the deductive database is changed, a maintenance system will be used to update the rule bank so that there will remain only those rules which respects the updates made to the deductive database.

References

[1] Weidong Chen : "Query Evaluation in Deductive Databases with Alternating Fixpoint Semantics", ACM Transactions on Database Systems vol. 20/3, 1995, pp. 239-287.
[2] Chun-San Hsu, Craig A. Knoblock: "Using inductive learning to generate rules for semantic query optimization", Advances in Knowledge Discovery and Data Mining, 1996, pp. 425-445.
[3] Haussler,D:"Quantifying Inductive Bias: AI Learning Algrithms and Valiant's Learning Framework.",Artificial Intelligence 36, 1988, pp. 177-221.
[4] Raghu Ramakrishnan:"Database Management Systems", WCB McGraw Hill, 1998.
[5] Jeffrey D. Ullman: "Principles of Databases and Knowledge-Base Systems, Vol I: Classical Database Systems", Computer Science Press, New York, 1988.
[6] Jeffrey D. Ullman: "Principles of Databases and Knowledge-Base Systems, Vol II: The new technologies", Computer Science Press, New York, 1989.
[7] S. Ceri, G. Gottlob, L. Tanca: "Logic Programming and Databases", Springer-Verlag Berlin Heidelberg 1990.
[8] L. Warshaw, Daniel P. Miranker: "Rule-Based Query Optimization, Revisited",CIKM, 1999, pp. 267-275

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

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLVIII**, Number 1, 2003

SCALABLE PLATFORM FOR MULTIMEDIA GROUP COMMUNICATION

PIROSKA HALLER

Abstract. The main aim of this paper is the development of a middleware platform, which will create the common abstracting level for different distributed media forming a communication group. The innovation of the presented platform consists in connecting two major aspects – the scalable architecture and the management of the quality parameters. In the same time the middleware platform has to be independent from the multimedia application that use it, and can run distributed on different platform.

We define an extended object model following the recommendation of the reference model, as basis for middleware service level. The proposed distributed and hierarchical architecture for different services offers more extensibility and efficiency to the application. But this architecture is part of the middleware and is transparent for the user level.

The proposed new architecture and measurement based management policies, make possible the continuous adaptation of the quality parameters to the modification of the requirement or the system resources at the application level, even if the network level not support guaranteed services.

Keywords: Distributed multimedia system, middleware platform, group communication, quality of services.

1. Introduction

The development of the interactive and distributed multimedia applications has introduced two new requirements, support for group communication and quality of services (QoS). The existing solutions on the level of the communication system cannot be adapted to this kind of multimedia applications. On the one hand the structure of the communication group may continuously modify, and the modifications of a terminal will affect all the members of the group. On the other hand the reservation-based quality of services will continuously block the unused resources, as it is very hard to estimate the requirements from the beginning. The advantage of the adaptive algorithms against of the algorithms based on reservation is evident.

Received by the editors: March 15, 2003.

2000 *Mathematics Subject Classification.* 68M14.

1998 *CR Categories and Descriptors.* D.2.11 [**Software engineering**]: Software architectures – *domain specific architectures* .

Presently the solutions follow two different directions. On the one hand the creation of several adaptive applications which would assure the transfer of the multimedia data even if the communication subsystem does not guarantee the quality of services[2,8], using different buffering and differentiated encoding techniques, but unfortunately they work on a relatively homogenous infrastructure. The other direction represents the extending of the communication subsystem with quality parameter maintaining services (Diffserv, Intserv), but this requires the modification of the infrastructure. Generally there are no integral solutions for maintaining the quality parameters for end-to-end connections on the application level. The effective usage of the services newly offered by the operation systems and by the communication subsystem is very difficult on the application level and it is necessary to create middleware platforms.

We suggest the creation of an middleware platform with a hierarchical and distributed structure, that will manage the communication groups for multimedia applications, providing the real time data access and the synchronous presentation of the data resulted from a distributed system. The proposed platform was implemented in collaboration with INTEGRASOFT in the HERMIX project.

The fundamental requirements that are impose for this platform represent the scalability of the system and the quality parameter management for end-to-end connections. The quality parameters management will control the distribution of the resources depending on the requirements, but based on globally optimal criteria, assuring in the same time the auto adaptation of the whole system at the modifications in a single node.

2. The Object Model

The computational point of view of the proposed platform corresponds to the RM-ODP[1] standard and defines the system as a set of location independent objects connected through explicit connections[4]. The objects can be accessed through the interfaces that can be message or stream type.

From engineering point of view the system may be considered as a set of terminal node objects forming communication groups. The terminal objects represent location independent entities that will be treated uniformly as encapsulated objects. These may vary in terms of granularity from a single communication media (for ex. text), to more types of communication media (for ex. video, audio, text, graphics, file), or to any number of identical media (for ex. 20 video streams). In fact a terminal node may be an interactive user presenting data required from the server, a file server, a database server, an audio or video broadcasting source, a monitoring point or any other combination of source and presentation objects.

The terminal objects represent a set of source objects, presentation objects, timer objects and a coordination object that is capable of generating, of processing and of interpreting scenarios[fig 1]. These scenarios represent temporal constrains between different types of data[7]. The terminal objects may implement two types

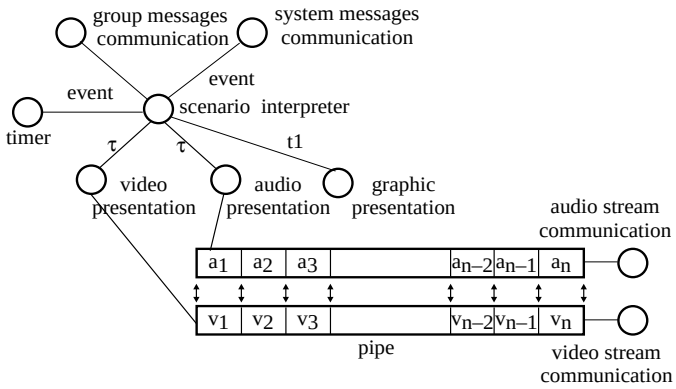


FIGURE 1. The structure of the terminal node

of communication: stream or message. The stream communication class may transmit notification events when receiving or transmitting a package based one the frame's time stamp, to the classes that have been registered for a certain event. Through this mechanism the synchronization or the presentation is supported according to certain scenarios. Each media will have its own communication object that will be dynamically created by the middleware platform.

Each terminal object has to contain message type communication objects representing the link to the middleware platform that provide the support for the management of the communication. The messages may be of different type: administration, system, error or group and will be transmitted asynchronously.

3. The Internal Structure

The middleware platform represents a set of services grouped in different modules: authentication, group management, quality parameters management, stream management.

The proposed platform has an open architecture, allowing for the integration of new services in case these correspond to the extended model. These modules may be created and dynamically loaded by the central manager object depending on the connection parameters specified in the terminal objects.

The central manager object is an object having a hierarchical structure, including different distributed active objects (group manager, terminal manager, stream manager, and QoS manager), communicating through messages[fig 2].

The active objects will include processing elements that support the parallel processing through the threads and will manage the communication channels for a subgroup of nodes, but only for a single media. Each active object will have a control object that will manage the rest and have a link to the hierarchically superior

active object. The management consists in the creation, suspending, resuming or destroying of threads according to the occurrence of an internal activation condition, with the received message or the notification event received. In case the resources are no longer sufficient or the capacity of a resource is exceed, it may initiate the creation of a new active object similar or hierarchically subordinated anywhere in the distributed system.

Each object will have a permanent connection with the hierarchically superior object where it will be registered together with the type of interface it implements. This hierarchically superior object will be polled in order to localize another active object to which it is to establish the connection. Thus there is the possibility that an object to be moved or replicated depending on the dynamic modification of the loading of the system. This hierarchical structure will be created according to the requirements of the infrastructure, but it will not be physically related to it. Unlike the models in which the controllers are to be distributed in the network nodes[9], the suggested system will be distributed depending on the load and the capacity of the resources, providing thus the scalability of the system. It will allow a best response time for the system, even if the interconnection system varies dynamically.

3.1. **Active objects.** Considering the requirements of the multi-point communication for the management of the quality parameters, for the real time synchronization and their insufficient implementation on the operating system level we propose the usage of the active objects for the creation and the controlling of the multi-point communication. In the followings we will define the internal behavior of the active objects that compose the middleware platform, their distribution and the relationship between them.

3.1.1. *Access manager.* The security services (authentication and authorization) will be handled by the access management object, which is a centralized object linked to a database and which will be created together with the central manager object wherever in the distributed system and with which communicate trough asynchronous messages. It is a two level security system, each group having attached a set of rights related to the administration of the group and to the access modes for different types of media, and similarly each registered terminal may have a set of rights. The module may use the security system of another platform or of the operating system. In case the access rights for the terminal are dynamically changed the access management object will send notification messages to the terminal manager.

3.1.2. *Terminal manager.* The central manager as a result of the connection request received from the terminal node dynamically creates the terminal manager and a message type bi-directional communication will be established with it for the transfer of the system messages. At the connection of a terminal the central

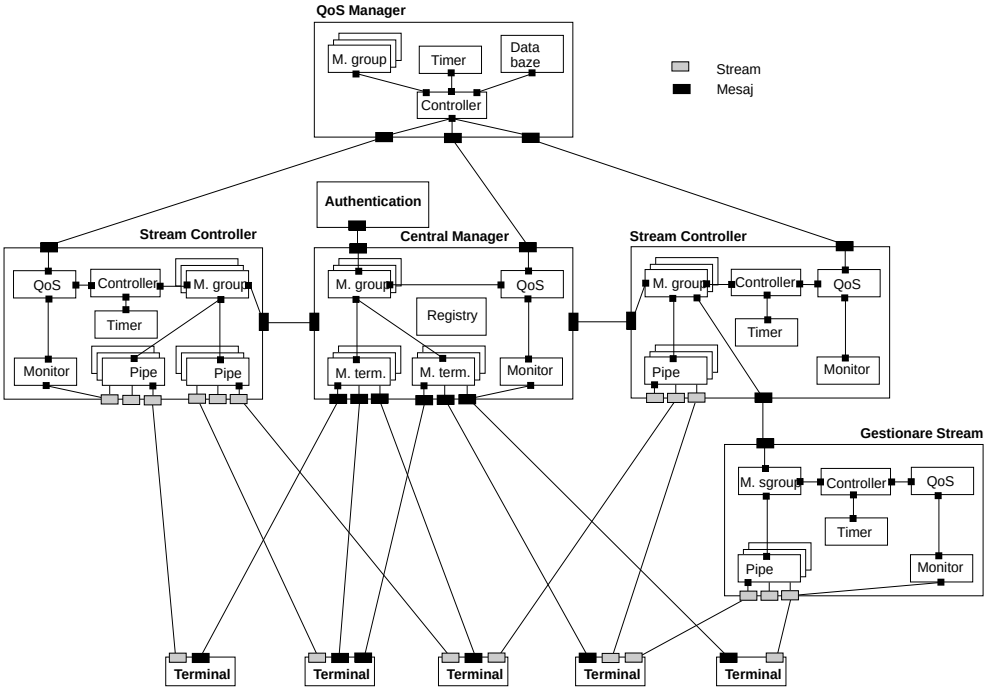


FIGURE 2. The middleware platform

manager object will initiate its authentication and all the data related to the rights and the terminal access will be stored locally in the terminal manager so that it should take over the verification task for the connected terminal. The terminal node will present a set of requested operations together with the requested transfer rate for these operations, and this value may be specified as an admission interval. The terminal manager will store the data related to the selected connection media together with the communication parameters. In case there are no explicit values specified for the quality parameters the terminal manager will generate these values based on certain qualitative options or priorities expressed by the terminal node and based on the monitored capacity. In order to implement the quality management system each channel will be characterized by the quality parameters such as the transfer rate, the delay, the jitter and the error rate. This transformation function of the quality parameters (linguistically expressed) in values or admissible intervals may influence the performance of the whole system.

3.1.3. *Group manager.* As the terminal nodes will be organized on discussion groups, for each of them a group manager object is created dynamically, that

will administer and will distribute the group messages as text, whiteboard and graphics, transmitted on connections different from the ones used by the system messages. The group manager will have separate data connections implemented through message communication objects for each terminal with traffic monitoring. Even for asynchronous message traffic there are prescribed transfer rates. The group manager will have handled the administration of the members of the list, respectively the verification functions of the rights for the group. It will communicate with the access manager object in order to obtain the list of rights and the eventual dynamical alterations that occur. It will send notification messages to the stream manager objects in case the structure of the group changes, in case a member is added to the group or is removed or in case a terminal changes its options related to the communication media.

3.1.4. *Stream manager.* The stream manager objects represent a set of special active objects handling the creation and the control of the stream data transfer among the terminals for a certain type of media.

The creation of different subtypes for different media is necessary as the control algorithms and the distribution of the data differs very much depending on the media type. In case the large number of communication groups requires it, more similar distributed stream manager object instances may be created, being registered at the central manager.

The stream manager object will have a bi-directional connection with each terminal implemented through stream communicators, through which it will receive data from the terminals and will distribute them to the members of the group. The data transfer has to respect the quality parameters established by local quality manager system, so that embedded controller and monitoring objects will be used. The timer objects will correlate the activity of the monitors and controllers, respectively will guarantee for the temporal constrains imposed for the continuous communication media[fig 2]. The controller objects will implement different techniques in order to reduce the sending rate depending on the attributes of the media, such as: filtering, reducing the frames, mixing, differentiated compressions. Each stream manager object is directly connected to the quality parameter manager object, where the monitoring data is regularly sent to and where the prescribed values are obtained for the data-receiving rate from the terminal nodes respectively for the data sending rates. The local quality parameter manager objects will handle the parameter administration inside a group, creating thus a hierarchical coordination structure. In the same time they will communicate with the quality parameters manager object, in order to send feedback messages in case the prescribed values cannot be satisfied with implemented algorithms. These messages initiate the re-negotiation of the prescribed values and the modification of the control algorithm parameters.

3.2. **Quality parameters management.** The architecture of the QoS management services includes the central QoS manager, the monitors and controllers distributed in the terminal objects and stream controllers. The role of the central manager unit is to determine the optimal requirement for the whole system, starting with the terminal's demand, but take in consideration the system constrains. This module monitoring the QoS parameters for the communication group based on messages received from the distributed QoS monitors across the communication units. The parameter value represents an average and takes in consideration in negotiation or renegotiations phases.

When a new terminal connecting for group send a messages with required QoS parameters and guaranteed capacities for all communication media. The parameters are defined at minimum and maximum values. Due of the policies implemented in the QoS manager, it renegotiates the required parameters with the new user of with the all members of the group in case of global QoS violation. When the user leave the group, or explicit change the required QoS the optimal value for QoS parameters for all media and for all members are recalculated.

Lets note with m the number of the terminals, with n the number of media, the output rate to send date from terminal j to media i with r_{ij}, and the input rate to receive date in node j to media i with q_{ij}. If U_j is the total guaranteed transfer capacities for the terminal node j, and G_i the total guaranteed transfer capacities for the stream controller i, we can write the following inequalities, that should be satisfied:

$$\sum_{i=1}^{n} r_{ij} + \sum_{i=1}^{n} q_{ij} \leq U_j, \quad j = \overline{1,m},$$
$$\sum_{j=1}^{m} r_{ij} + \sum_{j=1}^{m} q_{ij} \leq G_i \quad i = \overline{1,n} \quad (1)$$

We define the cost function, that minimize the difference between the prescribed input rate q_{ij} and the sum of the output rate q^*_{ij} from the other terminals (that affect also the delay, and the lost of the packets), respectively minimize the difference between the prescribed output rate r_{ij} and the required output rate r^*_{ij}. To obtain a more equilibrated solution we introduce the weight coefficients for the terms of the cost function. In this way we can define priorities in the system and can attach cost for some strident requirement. Lets note with $p_{ij}$the weight coefficients for input rate and with c_{ij} the weight coefficients for output rate.

$$f(r,q) = \sum_{i=1}^{n}\sum_{j=1}^{m} p_{ij}\left(q_{ij} - q^*_{ij}\right)^2 + \sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij}\left(r_{ij} - r^*_{ij}\right)^2 \quad (2)$$

The central QoS manager unit guarantees the global optimality and prevents the dominance of a group of terminals solving the quadratic optimization problem

minimizing the cost function (2) with inequality constrains (1)using inertia controlling methods [5]. When the structure of the group are changed, or the monitoring units detects change in the network capacities, the manager unit recalculate the optimal prescribed values. To prevent the system oscillation, the recalculation of the optimal parameters is initiated only if the modifications of the capacities are exceeding some threshold.

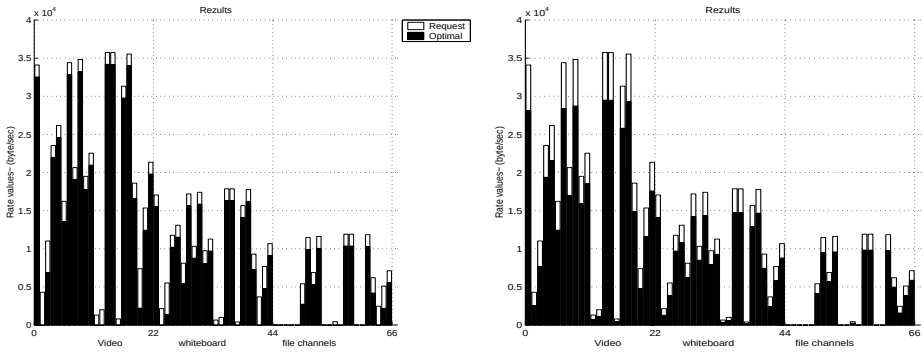
Using weight coefficients can obtain a more equilibrated requirement, at optimal cost. The weight coefficients can be defined by the user as extra cost to obtain the prescribed value near the required value, or can be generated by the manager unit according with the system loading. The user can modify the required values, or specify new priorities in system, or new qualitative option and the system should be adapt to the modification of the parameters.

We present some results, for one group with 22 users and 3 type of media (video/audio, shared graphics, file). Starting from the measured capacities and required transfer rate we calculate the optimal output transfer rate for each users on each media. The left diagram (figure 3) presents the results if the weight coefficients are equal. We can remarque the dominance of the high capacities channels. The slow channels can't send data, they only receive. Using the weight coefficients, where the costs are proportional with the capacities also the weak channels can send data (right diagram of figure 3). But the weight coefficient could be modified dynamically by the users with administrative rights.

The QoS manager unit offer a lot of services that accept the perceptive parameters like image dimension, image quality or audio quality level, and assign priorities for different media in accordance with the user choice. These qualitative parameters are transform in quantitative parameters need by the optimization problem. The introduction of the weight coefficient proportional with the costs in the criteria function allows for a better equilibrium of the system.

The calculated values will then be transmitted to the quality parameters manager objects distributed in the stream manager objects which will treat them as prescribed values and following it there will be selected the control algorithms corresponding to each local controller in part in order to maintain the rate and the delays in the prescribed limits.The proposed stream controller act as a classical numerical control unit, that generate an output at the required values, controlled the input and used the feedback values from the output. Under the assumption, that the network can not guarantee the required values for the delay, rate and jitter, and the operating system scheduling is non real-time, the presented solution assure the adaptation of the parameters in continuous way[6].

The most important role besides the controllers pertains to the monitoring objects that will be distributed in the terminal nodes as well. In case it is necessary the monitoring nodes may be distributed in the physical communication nodes as well, following the communication infrastructure.



user capacities in kbyte/sec:

822	172	441	806	810	649	834	809	835	780	805
52	80	837	836	6	818	836	744	296	614	810

FIGURE 3. The calculated optimal rate values

The monitoring of the system will include the classical communication parameters such as the transfer rate, delay, jitter, and the parameters specific to the active objects, such as the package loss due to the pipes dimension, compression factor, the temporary deviation from the prescribed value.

4. CONCLUSION

The proposed scalable architecture allows for the creation of certain application that will handle the connection of certain distributed terminal nodes – the number of which varies dynamically on a large scale, which will form the communication groups that will be configured and reconfigured dynamically. The management of the quality parameters will include the possibility to specify these requirements deterministically, probabilistically and stochastically. It will also include a support for the static and dynamical administration of the quality parameters, including the specification, admission negotiation and re-negotiation of the parameters, the controlling and the allocation of the resources, the monitoring of the system, independently of the degree it is guaranteed on the operating system's level.

On the other hand considering the platform as a classical distributed control system in which each type of media will be controlled by a controller object that will maintain the outgoing rate prescribed by the quality management system, controlling the incoming rate and monitoring the perturbations in the system, represents a very new approach. The quality management system will dynamically modify the temporal constrains for an object in order to maintain the optimality of the whole system, and in case it is necessary it will initiate the replication of

the respective object. The proposed mathematical model allows the continuous adapting of the parameters to the modifications of the system.

References

[1] G. Blair, J. B. Stefani: Open Distributed Processing and Multimedia. Addison Wesley, England, 1998.

[2] G. Coulson: A Distributed Object Platform Infrastructure for Multimedia Application. Computer Communications Vol 21, No 9, July 1998, 802–818, 1998.

[3] R. Davison, J. Hardwicke: A New Architecture for Open and Distributed Network Management. H. Zuidweg et al. (Eds.): IS&N'99, LNCS 1597, pp. 25-37. Springer-Verlag Berlin Heidelberg 1999.

[4] D. Duke, I. Herman, M. S. Marshall: PREMO A Framework for Multimedia Middleware, Specification, Rationale, and Java Binding. LNCS nr 1591, Springer Verlag 1999.

[5] P. Gill, W. Murray, M. Saunders, M. Wright: Inertia Controlling Methods for General Quadratic Programming. SIAM Review, vol 33, No. 1, 1–36, 1991.

[6] P. Haller: Contributii la implementarea sistemelor multimedia distribuite, PhD thesis University of Cluj-Napoca, 2002 Cluj-Napoca Romania.

[7] S. Ing, S. Rudkin: Simplifying Real-Time Multimedia Application Development Using Session Descriptions. H. Zuidweg et al. (Eds.): IS&N'99, LNCS 1597, pp. 305-314, Springer-Verlag Berlin Heidelberg 1999

[8] B. Li: AGILOS: A Middleware Control Architecture for Application-Aware Quality of Service Adaptations, phD thesis University of Illinois at UrbanaChampaign, 2000 Urbana, Illinois

[9] B. Li, W. Kalter, K. Nahrstedt. A Hierarchical Quality of Service Control Architecture for Configurable Multimedia Applications. in Journal of High-Speed Networks, Special Issue on QoS for Multimedia on the Internet, IOS Press, Vol. 9, pp. 153–174, 2000.

Petru Maior University Târgu-Mures, Romania, 1, Nicolae Iorga St., 4300 Târgu-Mures, Romania
40-265-211838
E-mail address: phaller@upm.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLVIII**, Number 1, 2003

THE LOAN-BANK CONTRACT: A SWAP OPTION

D. AKUME

Abstract. This review paper discusses financial options of the European type, particularly swap options and their pricing using the modified Black Model(1976). It also discusses the theory and modelling of *Contractual Saving for Housing*[1] as practiced by *German "Bauparkassen"*. After a discussion of both *Option Pricing* and *Loan Banking*, the last part of this paper reveals that a *loan-bank contract* is in actual fact a *financial option* on an *amortization swap*. The outlook aims at the valuation of such a *swaption*.

Keywords: Loan-bank, contract, option, swap, swaption.

1. Introduction

In this paper, we present a scientific perspective of *loan banking* by linking it to one of the cornerstones of modern Mathematical Finance, the theory of option pricing. The more seasoned users are fairly au fait with swaps and swaptions as forms of financial options[1].

The present survey paper was stimulated by the analysis of *loan banking*[25], done in collaboration between German "Bausparkassen" and the Center for Applied Computer Science, Cologne (ZAIK).

Firstly, we illustrate the mathematical valuation of swaps and swaptions following the modified Black Model[2] of 1976 for European options. Secondly, we review

Received by the editors: May 5, 2003.

2000 *Mathematics Subject Classification.* 91B28, 00A69, 46N10.

[1]A (Put)Call option (See Peter Reissner (1991), [24], page 23) gives its holder the right without obligation to (sell)purchase an underlying asset (to)from the writer at a predetermined price (strike price, K), on or before an agreed future date (expiry date T).

An American option is that which permits its holder to exercise the aforementioned right on or before the expiry date. Otherwise it is known as European option.

Boundary conditions for a European call (C_E) and a European put (P_E) option at expiry are as follows:

$$C_E = \max\{V_T - K; 0\} = (V_T - K, 0)_+$$
$$P_E = \max\{K - V_T; 0\} = (K - V_T, 0)_+,$$

whereby V_T is the spot price of the underlying asset at expiry date T.

[2]Here follows the Black formula (see also Biermann, [4], page 159):

$$\begin{aligned} \text{Call} &= B(0,T)[F \cdot N(d_1) - K \cdot N(d_2)] \\ \text{Put} &= B(0,T)[K \cdot N(-d_2) - F \cdot N(-d_1)] \end{aligned}$$

the theory and practice of Contractual Saving for Housing as practiced by German "Bausparkassen"[25], and present the model of a typical, discrete transaction. Finally, we analyse the *loan-bank contract*, compare it to a swaption and infer that it is a swaption on an amortization swap[3]

2. Interest Rate Swaps (IRS)

Definition 1. *An* interest rate swap (IRS[4]) *is a contractual agreement entered into between two counterparties under which they agree to exchange fixed for variable interest rates (mostly LIBOR*[5] *) periodically, for an agreed period of time based upon a notional amount of principal. The principal amount is notional because there is no need to exchange actual amounts of principal. Equally, however, a notional amount of principal is required in order to compute the actual cash amounts that will be periodically exchanged.*

2.1. **Concepts.** An IRS is an agreement of specified duration between two parties (Swap partners or Counterparties) for the exchange of interest rate payments relative to a nominal value (Notional Principal Amount) at predetermined periods of time. That is, counterparty A makes fixed interest payments to Counterparty B at specific time intervals. On the other hand, A receives from B variable payments relative to an agreed reference interest rate [6].

Counterparty B receives fixed interest payments (receiver position), That is to say, B pays variable. On the other hand, A makes fixed interest payments (payer position), That is, A receives variable payments in the swap. Variable and fixed coupon payments occur in predetermined time intervals. It is however stressed here that in practical terms, the dates of variable and fixed payments may not always coincide. If one denotes the variable cashflow interval[7] with Δt , the interest rate of reference would be agreed upon at time t, with the cashflow occurring at time $t + \Delta t$. Cashflows on both sides coinciding would mean the net value being paid to the beneficiary.

---

with

$$d_1 = \frac{\ln \frac{F}{K} + \frac{1}{2}\sigma^2 T}{\sigma\sqrt{T}}, \qquad d_2 = \frac{\ln \frac{F}{K} - \frac{1}{2}\sigma^2 T}{\sigma\sqrt{T}}, \tag{1}$$

whereby $N(.)$ is the gaussian distribution, $B(.,.)$ is the discount function and F is the future price (see also [19]) for details also of the Black/Scholes model.

[3]In actual fact a swaption, since it is an option on a future swap.

[4]In this paper the terms Interest Rate Swap and Swap are used interchangeably.

[5]London Interbank Offered Rate

[6]Rauleder (1993), [23], page 11.

[7]Rauleder (1993), [23], page 13: in the case of the variable cashflow interval being smaller than the fixed, fixed and variable cashflows will fall apart at certain dates.

2.2. **Application of Interest Rate Swaps.** Interest rate swaps have been widely used by the larger corporate institutions for some time as an efficient off-balance sheet method to manage interest rate exposure arising from their assets and liabilities. For example, a floating-rate borrower who expects a rise in interest rates can swap his floating rate obligation to a fixed rate obligation, thus locking in his future cost. Should he subsequently decide that rates have peaked, and that the trend is reversing, the interest obligation could be swapped back to a floating rate basis, thereby gaining advantage from the anticipated fall in rates.

Swaps are particularly useful in the restructuring of risk in an investment. Eventual interest rate risks can be hedged away with swaps. It is for this reason that swaps have become so important in financial management.

2.3. **Swap Pricing Model.** Pricing[8] a swap means determining the fixed interest rate R_{fix} of the swap (swap rate) such that, the value of the swap is zero at time $t = 0$.

It is clear from the definition that a swap is equivalent to a portfolio of two bonds, one short and the other long, one a fixed-rate bond and the other a floating rate bond[9].

Let $0 < t_1 < \cdots < t_n$ represent the reset dates of the swap.

The price of a Floating Rate Note (FRN) is always equal to the nominal value L[10] at time $t = 0$ irrespective of reset interval. Therefore the floating payment X_v at time $t = 0$ is the difference between L and the present value of the nominal value[11].

$$X_v = L - B(0, t_n)L = L\left(1 - B(0, t_n)\right),$$

whereby B(.,.) is the discount factor for the interval (t, T) as introduced in [19], page 2. Total fixed payment X_f at time $t = 0$ is a series of fixed payments at fixed interest rate R_{fix}

$$X_f = \sum_{i=1}^{n} B(0, t_i)R_{fix}(t_i - t_{i-1})L$$

The pricing of a swap is reduced to the problem of determining R_{fix}, such that the following equation holds:

$$\sum_{i=1}^{n} B(0, t_i)R_{fix}(t_i - t_{i-1})L = L\left(1 - B(0, t_n)\right)$$

---

[8]See B. Luderer; O. Zuschanke, [20], for a rigorous treatment of Swap-Pricing.

[9]Rauleder, [23], page 97; R. Kohn, [15].

[10]Rauleder (1993), [23], page 97.

[11]Unlike for a normal bond, the nominal value L is notional Capital, thus never actually changes hands.

3. Swaptions

A *swaption* is a combination of the following two financial instruments: *Interest Rate Swap (IRS)* and *Option*.

Swaptions first came into vogue in the mid-1980s in the US on the backof structured bonds tagged with a callable option issued by borrowers. With a callable bond, a borrower issues a fixed-rate bond which he may call at par from the investor at a specific date(s) in the future. In return for the issuer having the right to call the bond issue at par, investors are offered an enhanced yield. Bond issuers often issue an IRS in conjunction with the bond issue in order to change their interest profile from fixed to floating. Swaptions are then required by the issuer as protection to terminate all or part of the original IRS in the event of the bonds being put or called.

Definition 2. *A* Swaption[12] (Swap Option) *reserves the right for its holder to purchase a swap at a prescribed time and interest rate in the future (European Option).*

The holder of such a call option has the right, but not the obligation to pay fixed in exchange for variable interest rate. Therefore, this option is also known as "Payer Swaption". The holder of the equivalent put option has the right, but not the obligation to receive interest at a fixed rate (Receiver Swaption) and pay variable.

3.1. **Applications of Swaptions.** Actually, a swaption is an option on a forward interest rate. Like interest rate swaps, swaptions are used to mitigate the effects of unfavorable interest rate fluctuations at a future date. The premium paid by the holder of a swaption can more or less be considered as insurance against interest rate movements. In this way, businesses are able to guarantee risk limits in interest rates.

For instance, a five year swaption expiring in six months is the same as an option to contract a swap in six months time, and the swap will be valid for five years. To further buttress the point, an example is in order:

3.1.1. *Example:* Consider the case of a firm that will start servicing its debt six months from now. The debt is serviceable within five years, at a floating interest rate payable every six months.

This firm can protect itself against rising interest rates by purchasing a payer swaption. By paying a premium, the firm obtains the right to recive variable payments (mostly LIBOR) to pay a predetermined fixed interest rate eg. 12% p.a. for a five year period. The swap begins six months from now (expiry date of the Swaption).

There will be two possible outcomes at the expiry date:

[12]Peter Reissner (1991), [24], page 23.

(1) The market swap rates are higher than 12%: The option is exercised and its holder is able to satisfy his variable interest rate commitment at a rate below the market interest rate. Our firm thus gains.
(2) The market swap rate is below the strike rate: The swaption is not exercised and the firm turns to lower interest rates in the market.

3.2. **Pricing Swaptions With The Black-Model. Notation**

T	–	expiry date of Option
F	–	forward swap rate
$B(0,t_i)$	–	discount factor from date t_i, down to 0
X_v	–	variable interest rate
X_f	–	fixed interest rate
σ	–	volatility of swap
K	–	Strike rate
R_{fix}	–	Market swap rate at time T

The input parameter σ is obtained from market data[13].

Let $t_1 < \cdots < t_n$ represent the coupon dates for the swap and $t_0 = T$.

In deriving a pricing formula, we look at the swap underlying the swaption. The swap begins on the expiration date (T) of the swaption - this coincides with the first cashflows - and ends at time t_n. The swap comprises payments at floating interest rate X_v and payments at fixed interest rate X_f , relative to the n cashflows[14].

The variable interest rate payments are based on a benchmark (mostly LIBOR) at time t_k, a notional principal amount L and n interest periods $(t_i - t_{i-1})$

The fixed payments are based on the strike rate K, as well as same notional principal amount and periods.

At each date t_i, the interest rate R_{fix} shall be compared with the Strike rate K.

3.2.1. *Pricing Model for European Swaptions.* In case $R_{fix} > K$, the fixed interest payer gets paid a balance of $L \cdot (R_{fix} - K) \cdot (t_i - t_{i-1})$. Otherwise this balance goes to the variable interest payer.

The swap rate R_{fix} satisfies the following equation(see Section 2.3)

$$\sum_{i=1}^{n} B(T,t_i) R_{fix} (t_i - t_{i-1}) L = (1 - B(T,t_n)) L \tag{2}$$

[13][24], page 48 : historic or implicit Volatility.
[14]Robert V. Kohn, [15].

The left hand side of the above equation represents the value of the fixed payments at the rate of R_{fix} as of time T. The right hand side, on the other hand, represents the value of variable payments. Take note that the present value at variable interest rate will be equal to the notional principal amount.

The holder of an European swaption has the right to pay the fixed rate K and to receive a floating rate (payer swaption). In the case that $R_{fix} > K$, it means for the holder a value of

$$\begin{aligned} X_v - X_f &= (1 - B(T,t_n))L - \sum_{i=1}^{n} B(T,t_i)K(t_i - t_{i-1})L \\ &= \sum_{i=1}^{n} B(T,t_i)R_{fix}(t_i - t_{i-1})L - \sum_{i=1}^{n} B(T,t_i)K(t_i - t_{i-1})L \\ &= (R_{fix} - K)\sum_{i=1}^{n} B(T,t_i)(t_i - t_{i-1})L \end{aligned}$$

The ith term corresponds to the value of an European call option with expiry date T and coupon date t_i

$$L(t_i - t_{i-1})(R_{fix} - K)_+,$$

whereby R_{fix} is as introduced in section 2.3.

According to the Black Model (1976), the option value at time 0 as shown in footnote 1 is as follows:

$$B(0,t_i)(t_i - t_{i-1})L[F \cdot N(d_1) - K \cdot N(d_2)],$$

whereby F is the forward swap rate and

$$d_1 = \frac{\ln \frac{F}{K} + \frac{1}{2}\sigma^2 T}{\sigma\sqrt{T}}, \qquad d_2 = \frac{\ln \frac{F}{K} - \frac{1}{2}\sigma^2 T}{\sigma\sqrt{T}} = d_1 - \sigma\sqrt{T}.$$

d_1 and d_2 remain as in (1). The forward swap rate F is obtained from (2), by replacing $B(T,t_i)$ with $F(T,t_i) = \frac{B(0,t_i)}{B(0,T)}$.

The value of the swap C_E itself is the summation over all individual call options, i.e. over all i. And we get:

$$C_E = LA[FN(d_1) - KN(d_2)],$$

whereby

$$A = \sum_{i=1}^{n} B(0,t_i)(t_i - t_{i-1})$$

For an European receiver swaption in like manner, if $K > R_{fix}$ the the following holds:

$$\begin{aligned} X_f - X_v &= \sum_{i=1}^{n} B(T,t_i)K(t_i - t_{i-1})L - (1 - B(T,t_n))L \\ &= \sum_{i=1}^{n} B(T,t_i)K(t_i - t_{i-1})L - \sum_{i=1}^{n} B(T,t_i)R_{fix}(t_i - t_{i-1})L \\ &= (K - R_{fix})\sum_{i=1}^{n} B(T,t_i)(t_i - t_{i-1})L \end{aligned}$$

The *ith* term corresponds to the value of an European put option with expiry date T and coupon date t_i.

$$L(t_i - t_{i-1})(K - R_{fix})_+$$

According to the Black Model (1976), the option value at time 0 as shown in footnote 1 is as follows:

$$B(0,t_i)(t_i - t_{i-1})L[K \cdot N(-d_2) - F \cdot N(-d_1)],$$

whereby F, d_1, d_2 are as defined above, d_1 and d_2 are as introduced in (1). The forward swap rate F is obtained from (2) by substituting[15] $B(T,t_i)$ with $F(T,t_i) = \frac{B(0,t_i)}{B(0,T)}$.

The swap value P_E is the summation over all individual put options, i.e. over all i. And we get:

$$P_E = L \cdot A[K \cdot N(-d_2) - F \cdot N(-d_1)],$$

whereby

$$A = \sum_{i=1}^{n} B(0,t_i)(t_i - t_{i-1})$$

4. Loan Banking

4.1. **Concepts.** The idea behind loan banking can be illustrated with the following example:

Assuming that there are ten individuals each of who wants to build a house of the same size with none of them having sufficient capital to do so. If each of them saves ten percent of the required amount per year, each would be capable of building after ten years. However, if these individuals get together, the first person will already build after one year. In the subsequent nine years he will be busy amortizing his loan, so that the second person builds after two years etc. By so doing, the average waiting time to build is reduced from 10 to 5.5 years.

[15]Robert V. Kohn, [15].

The idea of getting together to form a *building cooperative*[16] is the basis upon which *loan banking* is founded.

Loan banking started in the United Kingdom in the late 18th. century in the form of closed cooperatives. These *Building Societies* as they were called had a limited number of members though.

4.2. **Contractual Saving for Housing.** Under this scheme, this loan bank offers loans to individuals and corporate bodies for the following purposes:

(1) construction and acquisition of a home
(2) renovation and completion of building projects
(3) purchase of building plots

However, to be eligible for such a lone, the aspirant has to open a savings account at *Loan Bank*. The Contractual Saving for Housing thus offers the owner of the contract a whole range of opportunities related to ownership of a real estate. The loan bank offers different types of contracts which generally evolve in the same trend.

The evolution of the contract can be classified into four phases - the contracting phase, the saving phase, the disbursement phase and the amortizing phase:

- **Contracting phase:**The owner signs the contract with the loan bank. The contract is a savings agreement for building purposes and locks conditions such as savings sum[17] and tariff[18]. The saving agreement also spells out the interest rate on saving and interest rate on loan. An inial deposit is paid in during this phase.
- **Saving phase:** The owner pays deposits in their saving account during this period. The goal of this phase is to fulfil the minimum requirements[19] that would make the contract eligible for the next phase (disbursement).
- **Disbursement phase:** Disbursement[20] takes place upon fulfillment of minimum requirements and on condition that the loan bank has the means[21] available.

[16]The terms building society, loan bank, building cooperative are used in this paper interchangeably.

[17]amount to be saved by customer plus loan to be obtained from building society as specified in the contract upon opening of account.

[18]Interest rates may differ depending on tariffs. Moreover, the fee charged upon conclusion of the contract, waiting period for contract maturity etc. also vary.

[19]As a rule, a minimum period of saving and a minimum amount to be saved are set in the loan agreement.

[20]The owner of the contract gets paid the contract sum (saving in account plus loan).

[21]Contracts are ranked and disbursed according to evaluation number. The evaluation number of a contract is an assessment of the intensity of saving with time $= \int_0^T \text{savings}(t)dt$, t being time.

The owner of a contract that is eligible for disbursement has the option of either accepting disbursement, deferring disbursement or terminating the contract (i.e. desists from taking loan)[22]. One of the reasons for terminating the contract at this stage could be the availability of a cheaper loan in the capital market.

- **Loan amortizing phase:** Systematic reimbursement of loan plus interest[23] levied on loan.

In an attempt here to model the aforementioned loan banking operations mathematically, a discrete model is chosen, given that in reality, financial transactions take place in discrete time steps [25].

Parameters for Building Society are an aggregate of those for individual contracts that make up the cooperative.

4.3. **Individual Contract Model.** Transactions involving a contract can be divided into four main phases as follows: Opening of account, saving, assignment, loan.

4.3.1. *Contracting and Saving.* Upon opening of an account, the contract is signed between the bank and the customer. This contract spells out contract conditions such as *contract volume tariff* etc.

The account once opened, the *saving phase* begins and money is subsequently saved in the account.

The following relationship exists for the savings S of customer i at time t :

$$S_i(t) = S_i(t-1) + P_i(t) + I_i(t),$$

whereby $S_i(t)$, for instance, denotes the cumulative Savings of customer or account i in year t. In the same logic, I is cumulative interest rate and P, cumulative deposit made in the account in the course of the year. The following relationship holds for the interest:

$$I_i(t) = \sum_{j=1}^{n} I_i(t_j)$$

$$I_i(t_j) = \frac{p}{n}\left(S_i(t-1) + \sum_{k=1}^{j} P_i(t_k)\right) \qquad j = 1, \ldots, n.$$

p is interest rate per annum and $t_1 < \cdots < t_n$ are discrete depositing periods within a year.

The saving phase of account i begins soon after opening of the account at time $t = t_{b,i}$ and ends upon assignment $t = t_{a,i}$. At time $t = t_{a,i}$, the owner of account i gets his entire saving plus loan paid out to him/her, except the owner decides to wait.

[22]The contract gets paid just its savings.

[23]The loan agreement contract already guarantees a fixed interest rate for the loan.

Generally, the following system of equations describes the saving phase (t stands for yearly periods, t_j stands for periods under one year, $t_{b,i}-1$ stands for the period prior to signing the contract):

$$S_i(t_{b,i}-1)=0$$

$$S_i(t)=S_i(t-1)(1+p)+\sum_{j=1}^{n}\left(P_i(t_j)+\sum_{k=1}^{j-1}\frac{p}{n}P_i(t_k)\right)$$

$$t=t_{b,i},\ldots,t_{a,i}$$

If at any point in time t_{close} a customer decides to terminate the contract, their saving is paid out to him/her.

4.3.2. *Disbursement.* an account that fulfils certain minimum requirements (is is going to be spelt out below) and whose eligibility coefficient lies above the target eligibility coefficient becomes eligible for disbursement. That is, such an account is paid back its savings plus loan amount. The payment of the loan for an account automatically causes the transition into the next phase i.e., the loan phase. The disbursement date, depending on the tariff in question, is also determined by the following three factors: The *minimum waiting period,* the *minimum saving coefficient* and the *minimum eligibility coefficient.*

- **Eligibility:** an assessment of the intensity of saving with time $=\int_0^T \text{savings}(t)dt$, t being time.
- **Savings coefficient:** savings in an account as a fraction of total nominal amount of accounts not yet approved for loan.
- **Waiting period:** After signing up with the loan bank, you are not eligible for a building loan until after saving a certain stipulated period of time .

4.3.3. *The Loan Phase.* This is the phase during which the loan is amortized. The net loan comprises the nominal amount minus saving. The net loan is paid out to the customer. This net loan is the amount to be amortized.

The customer pays a fixed amount, the amortizing amount (AA) at regular intervals (as agreed) to the cooperative. This so called amortization amount comprises the loan reimbursement plus interest on loan [24] (LI). This amounts to the following model for periods under one year t_j:

$$LI_i(t_j)=\frac{q}{n}\cdot Loan_i(t_{j-1}),$$

$$Loan_i(t_j)=Loan_i(t_{j-1})+LI_i(t_j)-AA_i(t_j),$$

[24]The method interest payment varies from tariff to tariff.

q being the *fixed* interest on loan. The remaining loan for each new period is obtained by subtracting the mount so far amortized from the foregoing loan plus the interest so far paid:

$$Loan_i(t) = (1 + \frac{q}{n})^n \cdot Loan_i(t-1) - \sum_{j=1}^{n} (1 + \frac{q}{n})^{n-j} AA_i(t_j).$$

The contract is automatically terminated when the entire loan is amortized.

4.4. **Aggregate Model.** The parameters required in the mathematical modelling of the building cooperative are outlined as follows:

- saving;
- loan;
- payment into account (during saving phase);
- interest and amortization payments (during loan phase);
- saving payout;
- loan payout;
- interest on saving;
- interest on loan.

With this in mind, a simple generalized model for a loan bank operating on a single tariff is set up. Let the number of customers be N, the number of periods in a year be n. The following relationships involving savings and loans is set up:

$$S(t) = \sum_{i=1}^{N} \left(S_i(t-1)(1+p) + \sum_{j=1}^{n} (P_i(t_j) + \sum_{k=1}^{j-1} \frac{p}{n} (P_i(t_k)) \right)$$

$$Loan(t) = \sum_{i=1}^{N} \left((1 + \frac{q}{n})^n \cdot Loan_i(t-1) - \sum_{i=1}^{n} \left((1 + \frac{q}{n})^{n-j} AA_i(t_j) \right) \right)$$

The savings and interest during saving phase is also modelled as follows:

$$S(t) = \sum_{i=1}^{N} S_i(t)$$

and

$$I(t) = \sum_{i=1}^{N} I_i(t)$$

The rest of the cooperative-wide aggregate parameters such as interest on savings and interest on loan are similarly computed from sum of individual accounts.

Here are a few more parameters that are required in modelling such a cooperative:

- level of fresh business;
- unassigned lot(liquidity);

- assigned lot;
- liquidity coefficient;
- available lot.

The unassigned lot ($UL(t)$) at time t is the overall saving Sin all accounts that are in the saving phase.

$$UL(t) = \sum_{i=1}^{N} S_i(t),$$

$$t_{b,i} \leq t \leq t_{a,i}.$$

The assigned lot ($AL(t)$) at time t is the overall sum of amounts in all accounts that have been assigned.

$$AL(t) = \sum_{i=1}^{N} NA_i(t),$$

$$t_{a,i} \leq t \leq t_{e,i}.$$

whereby $t_{e,i}$ is the final date.

Computing the target eligibility coefficient: A target eligibility coefficient is usually set for the period under consideration.

One however has to know first of all, how much resource is available for assignment. This so called *available lot* (AvL)is computed as a function of the savings and loan of that period. To the saving is added the deposit and in-coming interest payments that is expected upon assignment time. From the saving is also subtracted the saving payout as well as interest on saving. The loan is subtracted the amortization amount and to it is added loan payout.

$$AvL(t) = S(t) + P(t) - I(t) - Loan(t) + AA(t) - SP(t) - LoanP(t),$$

whereby $SP(t)$ stands for saving payout and $LoanP(t)$ for loan payout.

The target eligibility coefficient is determined, based on the available lot. All customers fulfilling the minimum requirements are sorted in descending order of eligibility coefficient. This list is assigned until the available lot gets finished. The eligibility coefficient of the last assigned account is the target eligibility coefficient.

In order to be able to follow-up cooperative development, the *liquidity ratio* ([16] and [17]) could be used to assess the development of the cooperative. certain parameters have been suggested in the past. The liquidity ratio represents the ratio of total loans to total saving.

$$\text{Liquidity ratio}(t) = \frac{Loan(t)}{S(t)}$$

It should be in the best interest of the cooperative to keep this ratio < 1, otherwise she would have to go borrowing. And as you know, borrowing is expensive.

5. The Loan Contract As Swaption

A contract that effectively gets disbursed at the disbursement phase does enter the loan phase and pays interest on the entire building loan at a fixed predetermined interest rate. During the loan phase, such a contract cannot take advantage of lower interest rates in the financial market without having to terminate the contract.

The fact that the customer may terminate the contract at the disbursement phase and desist from taking the loan, and instead obtain the loan from elsewhere at a lower rate, makes this contract similar to an *option* on a forward *interest rate swap* or *swaption* of the *European type*. Remember that the holder of a *swaption* has the right to choose between the market rate and the contractual interest rate at expiry[25]. Therefore this is an option on an amortizing swap.

Definition 3. *An amortizing swap is usually an interest rate swap in which the notional principal for the interest payments declines during the life of the swap.*

The notional principal amount in this case is the building loan in its amortizing phase. In the same vein, the expiry date of the option is the disbursement date of the loan contract. And there is an option on the contractual loan rate at the disbursement date.

6. Outlook

This author intends, in his future research, to set up a mathematical pricing model for forward amortization swaps as discussed above for loan-bank contracts.

References

[1] F. Black, *The Pricing of Commodity Contracts*, in Journal of Financial Economics,3, 1976, 167-179.
[2] F. Black, E. Derman and W. Toy, *A One-Factor Model of Interest Rates and its Application to Treasury Bond Options*, Financial Analysts Journal, Jan-Feb, 1990, 33-39.
[3] F. Black, M. Scholes, *The Pricing of Options and Corporate Liabilities*, in Journal of Political Economics,81, 1973, 637-654.
[4] B. Biermann, *Die Mathematik von Zinsinstumenten*, Oldenbourg Verlag, 1999, Mnchen.
[5] R.R.Jr. Bliss, E.I. Ronn, *Arbitrage-Based Estimation of Nonstationary Shifts in the Term Structure of Interest Rates*, Journal of Finance, 44, 1989 London, 591-610.
[6] M. Bs, *Optionsbewertung und Kapitalmarkt*, Verlag Josef Eul, Bergisch Gladbach/Kln, 1990.
[7] E. Crow, K.E. S Shimizu, *Lognormal Distributions, Theory and Applications*, Marcel Dekker Inc., 1988.
[8] F. Fabozzi, *Bond Markets, Analysis and Strategies*, Prentice-Hall, New Jersey, 1989.
[9] B. Ganter, R. Wille, *Formale Begriffsanalyse: Mathematische Grundlagen*, Springer, Berlin, Heidelberg, 1996.
[10] F. Heitmann, *Bewertung von Zinsfutures*, Diplomarbeit am Instutut fr Entscheidungstheorie und Unternehmensforschung, Universitt Karlsruhe(TH), Januar (1992), Fritz Knapp Verlag, Frankfurt am Main 1992.

[25]The expiry date in this case would be the the date of disbursement.

[11] T.S.Y. Ho, S.B. Lee, *Term Structure Movements and Pricing Interest Rate Contingent Claims*, Journal of Finance, 41, 1986, 1011-1029.
[12] J. Hull, *Options,Futures and Other Derivative Securities*, Englewood Cliffs, New Jersey:Prentice Hall 1989.
[13] J. Hull, A. White, *On Derivatives*, Risk Publications, London, 1996.
[14] B. Knab, R. Schrader, I. Weber, K. Weinbrecht, B. Wichern *Mesoskopisches Simulationsmodell zur Kollektivfortschreibung*, Center for Applied Computer Science, Report 97.295, 1997.
[15] R.V. Kohn, *Derivative Securities - Section 11*, `http://www.math.nyu.edu/faculty/kohn/derivative.securities/section11.pdf`
[16] H. Laux, *Entwicklung der Bauspartechnischen Kennzahlen bei den privaten und den oeffentlich-rechtlichen Bausparkassen bis 1989*, in: Blaetter der Deustchen Gesellschaft fuer Versicherungsmathematik e.V., XVI.1(1): Pages 37 - 60, April 1991
[17] H. Laux, *Verlauf der Bauspartechnischen Kennzahlen in an- und auslaufenden Tarifbestaenden des Bausparens*, in: Blaetter der Deustchen Gesellschaft fuer Versicherungsmathematik e.V., XVI.3(3): Pages 365 - 371, April 1992
[18] W. Lehmann, *Die Bausparkassen*, Fritz Knapp Verlag, Frankfurt am Main, 1965.
[19] B. Luderer, D. Akume *Einige Aspekte der Bewertung von Swaptions*, Technische Universitaet Chemnitz, Fakultaet fuer Mathematik, Preprint 2001-15.
[20] B. Luderer, O. Zuschanke, *Ein einheitlicher Zugang zum Pricing von Swaps*, Preprint 2000-14, TU Chemnitz.
[21] R.F.M.L. Obermann, *Zinsrisikopotential-Kennziffer zur Quantifizierung des Zinsrisikos von Zinsswaps, -Futures und -Optionen*, Fritz Knapp, Frankfurt am Main, 1990.
[22] Z.E. Prisman, *Pricing Derivative Securities*, Academic Press, 2000.
[23] R. Rauleder, *Bewertung, Anwendungsmglichkeiten und Hedginsstrategien von Swaptions*, Fritz Knapp Verlag, Frankfurt am Main, 1994.
[24] P. Reiner, *Zur analytischen Bewertung von Zinsoptionen*, Verlag Peter Lang GmbH, Frankfurt/Main, 1991.
[25] I.M. Vannahme, *Clusteralgorithmen zur mathematischen Simulation von Bausparkollektiven*, doctoral thesis, University of Cologne, Cologne, 1996.
[26] G.-W. Weber, *Mathematische Optimierung in Finanzwirtschaft und Risikomanagement - diskrete, stetige und stochastische Optimierung bei Lebensversicherungen, Bausparvertrgen und Portfolios*, lecture held at Chemnitz University of Technology, summer semester 2001.
[27] P. Wilmott, S. Howison, J. Dewynne, *The Mathematics of Financial Derivatives*, Cambridge, 1995.
[28] P. Wilmott, *Week 9: The Black-Scholes Solution And The "Greeks"* , `http://www.ph.qmw.ac.uk/∼oleg/Week9.pdf`

Computer Science Department, University of Buea, P.O. Box 63, Buea, Cameroon
E-mail address: `daniel.akume@minesup.gov.cm`

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLVIII**, Number 1, 2003

STOCHASTIC OPTIMIZATION OF QUERYING DISTRIBUTED DATABASES I. THEORY OF FOUR RELATIONS JOIN

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

Abstract. Stochastic query optimization problem for multiple join is addressed. In Part I two sites model of Drenick and Smith (1993) is extended to four relations stored at four different sites. Our model leads to a special kind of nonlinear optimization problem (P). It is proved (Theorem 5.1) that this problem has at least one solution. In Part II an ad hoc constructive model for solving problem (P) is proposed. In Part III a new evolutionary technique is used for solving problem (P). Results obtained by the two considered optimization approaches are compared.

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

1. Introduction

The ability of distributed systems for concurrent processing motivates the distribution of a database in a network. The query optimization problem for a single query in a distributed database system was treated in great detail in the literature. Many algorithms were elaborated for minimizing the costs necessary to perform a single, isolated query in a distributed database system. Some methods can be found in Özsu and Valduriez (1999), Date (2000). Most approaches look for a deterministic strategy assigning the component joins of a relational query to the processors of a network that can execute the join efficiently and determine an efficient strategy for the data transferring.

A distributed system can receives different types of queries and processes them at the same time. Query processing strategies may be distributed over the processors of a network as probability distributions. In this case the determination of the optimal query processing strategy is a stochastic optimization problem. There is a different approach to query optimization if the system is viewed as one which receives different types of queries at different times and processes more than one query at the same time.

Received by the editors: May 15, 2003.

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

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

The multiple-query problem is not deterministic; the multiple-query-input stream constitutes a stochastic process. The strategy for executing the multiple-query is distributed over the sites of the network as a probability distribution. The "decision variables" of the stochastic query optimization problem are the probabilities that component operators of the query are executed at particular sites of the network.

Drenick and Smith (1993) extend the state-transition model proposed by Lafortune and Wong (1986) and the original multiprocessing model (see Drenick and Drenick, 1987, Drenick, 1986). The main objective of the state-transition model is to give globally optimal query-processing strategies. Drenick and Smith (1993) treat the single-join model, the general model for the join of two relations and a multiple-join with three relations, which are stored at two different sites. The stochastic model for the join of three relations, which are stored at three different sites is presented in Varga (1998) and Varga (1999).

Stochastic query optimization problem leads to a nonlinear programming problem, which is a specific one. General models of sequential and parallel operation for the specified type queries are treated in Varga (1999). Stochastic query optimization model using semijoins is presented in Markus, Morosan and Varga (2001).

The aim of this paper is to extend the stochastic model to the join of four relations. In Section 2 the case when the relations are stored at four sites is considered. The stochastic query optimization problem in case of four relations leads to a constrained nonlinear optimization problem. Considering the complexity of obtained nonlinear problem two complementary methods for solving this problem are proposed. Theorem 5.1 proves, that the nonlinear optimization problem has at least one solution. In Part II of the paper a constructive method for solving the nonlinear programming problem is given.

Due to the successfully application in the recent past of the evolutionary algorithms for solving very difficult optimization problems evolutionary methods seem to be quite appealing for solving our optimization problem. We will consider evolutionary techniques based on a dynamic representation (Dumitrescu, Grosan and Oltean, 2001, Grosan and Dumitrescu, 2002). This technique called *Adaptive Representation Evolutionary Algorithm* (AREA) is described in Part III. The results obtained by applying these different approaches are presented in Part III. Two sets of values for constants are used in these experiments. Solutions are nearly the same. The CPU time required for solving the optimization problem by using evolutionary algorithm is less than the CPU time required by the constructive method.

2. Four relations join

Consider four relations stored in different sites of the distributed database. The join of these four relations will be defined in the context of stochastic model of

Drenick and Smith (1993). Consider relations A, B, C, D stored at the sites 1,2,3 and 4 respectively.

Denote by Q_4 the single-query type consisting of the join of four relations:

$$Q_4 = A \bowtie B \bowtie C \bowtie D.$$

Initial state of relations referenced by the query Q_4 in the four-site network is the column vector defined as:

$$s_0 = \begin{pmatrix} A \\ B \\ C \\ D \end{pmatrix}$$

where the i-th component of the vector s_0 is the set of relations stored at site i, $i \in \{1, 2, 3, 4\}$ at time $t = 0$.

Initial state s_0 is given with time-invariant probability

$$p_0 = p(s_0)$$

i.e. p_0 is the probability that relation A is available at site 1, relation B at site 2, relation C at site 3, and relation D at site 4. The four relations are not locked for updating or are unavailable for query processing for any other reason. We assume that the input to the system consists of a single stream of type Q_4.

For the purpose of stochastic query optimization we enumerate all logically valid joins in the order in which they may be executed. Let us suppose that Q_4 has three valid execution sequences:

$$Q_4S_1 = (((A \bowtie B) \bowtie C) \bowtie D),$$

$$Q_4S_2 = ((A \bowtie B) \bowtie (C \bowtie D)),$$

$$Q_4S_3 = (A \bowtie (B \bowtie (C \bowtie D))).$$

Sequence Q_4S_1 can be applied if

$$A \cap B \neq \emptyset.$$

So the join

$$B' = A \bowtie B$$

is executed before the join

$$C' = B' \bowtie C.$$

The last executed join will be

$$D' = C' \bowtie D.$$

The sequence Q_4S_2 is adequate for parallel execution.

3. Stochastic query optimization model

The system that undergoes transition in order to execute the join of four relations is described in this section as in Drenick and Smith (1993). The strategy for executing the multiple join is distributed over the sites of the network. Conditional probabilities are associated with the edges of the state-transition graph. Executing a multiple join is equivalent to solve a optimization problem. This problem is referred as *stochastic query optimization model*. Theorem 3.1 states, that the stochastic query optimization model for the multiple join query defines a nonlinear optimization problem.

We exemplify with the execution of the join Q_4S_1.

The state-transition graph for sequence Q_4S_1 is given in Figure 1. For one state of the state-transition graph the i^{th} line contains the relations stored at site i. We will associate a transition probability to each transition arc of the state-transition model. Let p_{ij} denote the conditional, time-invariant probability that the system undergoes transition from state s_i to state s_j. Given the initial state s_0, we can execute the first step of Q_4S_1transferring relation B from site 2 to site 1, or transferring relation A from site 1 to site 2.

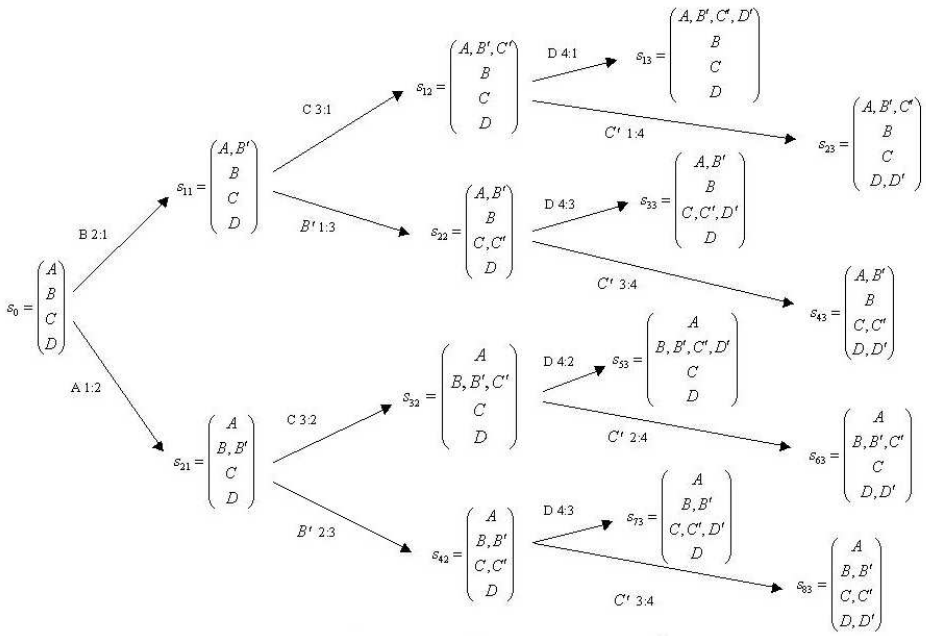


Figure 1. State-transition graph for the join Q_4S_1 of four relations

States of the transition graph are labeled as s_{ij}, where j is the step in computing the multiple join and$i$ denote the number of the selected strategy within the step.

Using the first strategy the system undergoes transition from state s_0 to the state s_{11} with probability $p_{0,11}$. The system may choose the second strategy with probability $p_{0,21}$ when the system undergoes transition to state s_{21}.

In order to compute

$$C' = B' \bowtie C,$$

if the system is in state s_{11} relation B' may be transferred from site 1 to site 3 or relation C from site 3 to site 1 and similar for the other states of the state-transition graph.

With respect to the stochastic query optimization model we can state the following Theorem.

Theorem 3.1 The stochastic query optimization model for the multiple join query of type Q_4 defines a nonlinear optimization problem.

Proof: We will associate the join-processing times with the nodes of the state-transition graph and communication times to the arcs of the graph. Let $T_i(X)$ denote the total processing time required for computing in state i.

So we have:

$$\begin{aligned}
&T_{11}(B') = c_{21}\ (B) + t_1\ (A \bowtie B),\\
&T_{12}(C') = c_{31}\ (C) + t_1\ (B' \bowtie C),\\
&T_{32}(C') = c_{32}\ (C) + t_2\ (B' \bowtie C),\\
&T_{13}(D') = c_{41}\ (D) + t_1\ (C' \bowtie D),\\
&T_{33}(D') = c_{43}\ (D) + t_3\ (C' \bowtie D),\\
&T_{53}(D') = c_{42}\ (D) + t_2\ (C' \bowtie D),\\
&T_{73}(D') = c_{43}\ (D) + t_3\ (C' \bowtie D),\\
&T_{21}(B') = c_{12}\ (A) + t_2\ (A \bowtie B),\\
&T_{22}(C') = c_{13}\ (B') + t_3\ (B' \bowtie C),\\
&T_{42}(C') = c_{23}\ (B') + t_3\ (B' \bowtie C),\\
&T_{23}(D') = c_{14}\ (C') + t_4\ (C' \bowtie D),\\
&T_{43}(D') = c_{34}\ (C') + t_4\ (C' \bowtie D),\\
&T_{63}(D') = c_{24}\ (C') + t_2\ (C' \bowtie D),\\
&T_{83}(D') = c_{34}\ (C') + t_4\ (C' \bowtie D).
\end{aligned} \tag{3.1}$$

Denote by $c_{ij}(R)$ the time required to transfer the relation R from site i to site j. $t_i\ (E)$ denotes the necessary time to calculate the expression E in the site i. The expected delay due to computing the join is the product of the delay and the corresponding transition probability. The mean processing time τ_i at site i can be obtained by summing for each state for which there is something to work in the site i, the product of the necessary time for processing multiplied by the probability that the system is in the corresponding state.

Let us suppose that input queries of type Q_4 arrive at the system at average intervals of length δ and successive inputs are statistically independent. It is reasonable to require that none of the processors in the network be allowed to

take longer on the average than the period δ to execute its task. If it did, the cumulative delay at each site could increase indefinitely due to queuing, requiring infinite buffer storage at each site. The system may be regarded as overloaded if the mean processing time τ_i is permitted to exceed δ at any site.

Such overload can be avoided if the following inequalities hold:

$$\tau_i \leq \Delta < \delta,$$

where Δ represents a common upper bound on τ_i for each processor i in the network.

In order to maximize the system *query-processing capacity*

$$\lambda = \frac{1}{\delta}$$

the system's mean interarrival time Δ may be minimized, where

$$(\delta - \Delta) > 0,$$

is chosen sufficiently large to provide adequate buffer storage requirements.

The mean processing times $\tau_i, i = 1, 2, 3, 4$ are expressed as:

$$\begin{aligned}
\tau_1 &= T_{11}(B')p_{0,11} + T_{12}(C')p_{0,11}p_{11,12} + T_{13}(D')p_{0,11}p_{11,12}p_{12,13}, \\
\tau_2 &= T_{21}(B')p_{0,21} + T_{32}(C')p_{0,21}p_{21,32} + T_{53}(D')p_{0,21}p_{21,32}p_{32,53} \\
\tau_3 &= T_{22}(C')p_{0,11}p_{11,22} + T_{42}(C')p_{0,21}p_{21,42} + T_{33}(D')p_{0,11}p_{11,22}p_{22,33} \\
&\quad + T_{73}(D')p_{0,21}p_{21,42}p_{42,73}, \\
\tau_4 &= T_{23}(D')p_{0,11}p_{11,12}p_{12,23} + T_{43}(D')p_{0,11}p_{11,22}p_{22,43} \\
&\quad + T_{63}(D')p_{0,21}p_{21,32}p_{32,63} + T_{83}(D')p_{0,21}p_{21,42}p_{42,83}.
\end{aligned} \tag{3.2}$$

Therefore the stochastic query optimization problem for the query Q_4S_1 is given by:

$$(P_1)\begin{cases}
\text{minimize } \Delta_1 \\
\quad \text{subject to:} \\
\tau_i \leq \Delta_1, i = 1, 2, 3, 4 \\
p_{0,11} + p_{0,21} = 1, \\
p_{11,12} + p_{11,22} = 1, \\
p_{21,32} + p_{21,42} = 1, \\
p_{12,13} + p_{12,23} = 1, \\
p_{22,33} + p_{22,43} = 1, \\
p_{32,53} + p_{32,63} = 1, \\
p_{42,73} + p_{42,83} = 1, \\
p_{0,11}, p_{0,21}, p_{11,12}, p_{11,22}, p_{21,32}, p_{21,42}, p_{12,13} \in [0, 1], \\
p_{12,23}, p_{22,33}, p_{22,43}, p_{32,53}, p_{32,63}, p_{42,73}, p_{42,83} \in [0, 1].
\end{cases}$$

This concludes the proof.

The obtained problem (P_1) is a constrained nonlinear optimization problem. In the next section we propose a constructive approach for solving the optimization problem (P_1).

4. Stochastic query optimization problem

Let us consider the following notations:

$$\begin{aligned}
&h_1(z_1, z_2, \ldots, z_{14}) = c_1z_1 + c_2z_1z_3 + c_3z_1z_3z_7,\\
&h_2(z_1, z_2, \ldots, z_{14}) = c_4z_2 + c_5z_2z_5 + c_6z_2z_5z_{11},\\
&h_3(z_1, z_2, \ldots, z_{14}) = c_7z_1z_4 + c_8z_2z_6 + c_9z_1z_4z_9 + c_{10}z_2z_6z_{13},\\
&h_4(z_1, z_2, \ldots, z_{14}) = c_{11}z_1z_3z_8 + c_{12}z_1z_4z_{10} + c_{13}z_2z_5z_{12} + c_{14}z_2z_6z_{14},
\end{aligned} \tag{4.1}$$

where $z_1 = p_{0,11}$,

$z_2 = p_{0,21}$,

$z_3 = p_{11,12}$,

$z_4 = p_{11,22}$,

$z_5 = p_{21,32}$,

$z_6 = p_{21,42}$,

$z_7 = p_{12,13}$,

$z_8 = p_{12,23}$,

$z_9 = p_{22,33}$,

$z_{10} = p_{22,43}$,

$z_{11} = p_{32,53}$,

$z_{12} = p_{32,63}$,

$z_{13} = p_{42,73}$,

$z_{14} = p_{42,83}$.

Expressing z_{2k-1}, $k = 1, 2, \ldots, 7$, from equality restrictions of problem (P_1) we have:

$$z_{2k-1} = 1 - z_{2k}.$$

By replacing z_{2k-1}, $k = 1, 2, \ldots, 7$, in the inequalities of (P_1)

$$\tau_i \le \Delta_i,$$

the problem (P_1) can be rewritten as the next optimization problem (P_2):

$$(P_2)\begin{cases}
\text{minimize } \Delta_1\\
\quad \text{subject to:}\\
f_1(x_1, x_2, \ldots, x_7) \le \Delta_1,\\
f_2(x_1, x_2, \ldots, x_7) \le \Delta_1,\\
f_3(x_1, x_2, \ldots, x_7) \le \Delta_1,\\
f_4(x_1, x_2, \ldots, x_7) \le \Delta_1,\\
x_1, x_2, \ldots, x_7 \in [0, 1]\,.
\end{cases}$$

The number of relations and sites in one distributed database can be different. Resulting nonlinear optimization problem has different number of variables and constraints. Therefore we have to generalize problem (P_2) for an arbitrary number of relations and sites.

Let us consider p continous functions

$$f_1, ..., f_p : [0,1]^n \to R_+,$$

where p is the number of sites in the distributed database and $f_i, (i = 1, \ldots, p)$ represents the mean processing time at site i.

Our optimization problem (P_2) may be generalized to the following optimization problem (P_p).

$$(P_p) \begin{cases} \text{minimize } \Delta_1 \\ \quad \text{subject to:} \\ f_1(x_1, x_2, \ldots, x_n) \leq \Delta_1, \\ \vdots \\ f_p(x_1, x_2, \ldots, x_n) \leq \Delta_1, \\ x_1, x_2, \ldots, x_n \in [0,1]\,. \end{cases}$$

5. General optimization framework

In this section problem (P_p) is considered as an instance of a more general framework. The new framework is necessary for establishing conditions under which problem (P_p) has a solution.

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

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

be continuous strictly positive functions.

Consider the next generic optimization problem:

$$(P) \begin{cases} \text{minimize } y, y \in R \\ \quad \text{subject to:} \\ x \in X, (X \text{ is a compact metric space}), \\ y > 0, \\ f_1(x) \leq y, \\ \vdots \\ f_p(x) \leq y. \end{cases}$$

With respect to problem (P) we can state the following Theorem. For proving it some concepts and results are needed (see for instance Rudin, 1976).

Theorem 5.1: Problem (P) has at least one solution.

Proof. Let X is be compact metric space and $f : X \to R$ be the function defined as

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

Since function f is continuous and X is a compact metric space, according to the Weierstrass theorem, there exists a point $x_0 \in X$ such that x_0 is the global minimum of the function f, i.e.

$$f(x_0) = \min_{x \in X} f(x).$$

We have to prove that

$$f(x_0) = \min y.$$

Let us suppose that it exists $y_0 \in R_+^*$ such that

$$f(x_0) > y_0,$$

and y_0 satisfies the inequalities from the problem (P) for $x_0^* \in X$, i.e.:

$$\begin{aligned} f_1(x_0^*) &\leq y_0 \\ &\vdots \\ f_p(x_0^*) &\leq y_0. \end{aligned}$$

From these inequalities we obtain

$$\begin{aligned} f(x_0^*) &= \max\{f_1(x_0^*), ..., f_p(x_0^*)\} \\ &\leq y_0 \\ &< f(x_0). \end{aligned}$$

But this contradicts the assumption that $x_0 \in X$ is the global minimum of the function f. Therefore the assumption concerning the existence of a value y_0 such that

$$f(x_0) > y_0$$

is false. This completes the proof. □

6. Conclusions

Stochastic optimization model of querying distributed databases, presented by Drenick and Smith (1993), is extended to the join of four relations. These four relations are stored in four different sites. Theorem 3.1 states, that the stochastic query optimization problem in case of four relations leads to a constrained nonlinear programming problem. The problem of querying the distributed database is generalized for p sites. General constrained nonlinear problem (P) is formulated. Theorem 5.1 proofs that problem (P) has at least one solution.

Acknowledgments: We are grateful to professors Cs. Varga and M. Frentiu for their valuable suggestions and to the Journal's anonymous reviewers for useful observations.

References

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

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

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

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

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

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

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

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

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

[10] M. T. Özsu, P. Valduriez (1991) : *Principles of Distributed Database Systems*, Prentice-Hall.

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

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

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

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

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

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

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

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

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

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLVIII**, Number 1, 2003

A DESIGN PROPOSAL FOR AN OBJECT ORIENTED ALGEBRAIC LIBRARY

VIRGINIA NICULESCU

Abstract. Object oriented programming and design patterns introduce a high level of abstraction that allows us to implement and work with mathematical abstractions. Classic algebraic libraries, based on imperative programming, contain subalgorithms for working with polynomials, matrices, vectors, etc. Their big inconvenience is the dependency on types. For example, a polynomial can be built over any kind of algebraic unitary commutative ring $(R, +, *)$, and we have to define a different set of procedures that implement the common operations with polynomials, for every such ring.

We propose here an object oriented approach for designing an algebraic library, based on design patterns, which remove this inconvenient. The big advantage of this approach is given by the creational design patterns, specifically Abstract Factory and Singleton. They introduce significant flexibility and abstractness. Thus, we may work with abstract algebraic structures, such as: groups, rings, fields, etc., like mathematicians do.

Keywords: OOP, design, patterns, algebraic structures, abstracness

1. Introduction

During the time, many algebraic libraries, which contain subalgorithms for working with polynomials, matrices, vectors, etc., have been built [7, 8, 2].

The big inconvenience of classic imperative algebraic libraries is their dependency on the types. For example, an polynomial can be built over any kind of algebraic unitary commutative ring $(R, +, *)$, and we have to define a different set of procedures that implement the common operations with polynomials, for every such ring.

Some other approaches are based on generic programming[8]. This may represent a solution but it has some inconveniences:

Received by the editors: May 20, 2003.

2000 *Mathematics Subject Classification.* 98A70,11C08,11C20.

1998 *CR Categories and Descriptors.* D.1.5 [**Software**]: Programming Techniques – *Object-oriented Programming*; D.3.3 [**Software**]: Programming Languages – *Language Constructs and Features*; E.2 [**Data**]: Data Storage Representations.

- Not all object oriented languages have mechanisms for genericity, and those having these mechanisms – like STL of C++ – don't offer the possibility to constrain the parameterized types to any explicit conditions. (Still, in the "Generic Java" proposals, the parametric types may be constrained to some conditions[10], but GJ is yet not used.)
- A parameterized matrix multiplication routine could be written and instantiated for matrices over integers, rationals, maybe real and complex numbers, numbers in Z/mZ and so on. But, this will produce the complete multiplication code for each type, in the executable.

Object oriented programming and design patterns form a very good framework for implementing a general algebraic library. We analyze here an object oriented approach for designing an algebraic library, based on design patterns, which remove the inconvenient of type dependency. We will create abstract classes that implement general abstract algebraic structures, and we will use Abstract Factory design pattern for building the special values.

Object oriented programming has been used before for designing some algebraic libraries [8], but the difference is given by the usage of creational design patterns. In this way, we can build not only a flexible numerical algebraic library, but a general abstract algebraic library.

2. Creational Design Patterns

Creational design patterns abstract the instantiation process. They are based on composition and inheritance. They allow us to make the pass from the hardcoding of a fixed set of behaviors towards defining a smaller set of fundamental behaviors that can be composed into any number of more complex ones. Thus, creating objects with particular behaviors requires more than simply instantiating a class. Five creational design patterns are considered to be classic: Abstract Factory, Prototype, Factory Method, Builder and Singleton [5].

We need, for our library, to use some special values, such as null and unity elements, in some classes where we don't know the concrete type of the special values. So, we have to create them by using special methods.

For example, building a general class for a polynomial does not have to be dependent on the coefficient types. So, we will use a general abstract type for the coefficients. But for the implementation of the class polynomial, we need to work with the special values 0 and 1.

Abstract Factory, Factory Method, and Prototype design pattern may be used for our purpose, and we analyse which one is more appropriate.

Factory Method is not appropriate for building the library, because it imposes the derivation of new classes for the main algebraic structures. For example, for a

polynomial, the purpose is to define in our library a completely defined class, and let the user to use it for any appropriate coefficient type.

The Prototype pattern may be used with some advantages. The Prototype pattern imposes only that every type defines a method **clone**, that allow an element to be copied. The advantage is that using the Prototype pattern leads to fewer classes than using the Abstract Factory pattern, but the structuring of the library would not be so good.

So, we have chosen the Abstract Factory design pattern. Different factory classes that define creational methods for the special values are defined, such as: `GroupFactory, FieldFactory`, etc.

Factory classes may use Singleton pattern, in order to allow a single factory instance for each type.

3. The Design Scheme

We start from a general algebraic element: `AlgElem`, which is implemented as an interface with no methods. We define the interfaces corresponding to algebraic elements, starting from their definition.

3.1. **Basic structures.** A very well known and used basic algebraic structure is the group. We define consequently an interface `GroupElem` that extends `AlgElem` (Figure 1). This interface contains a method `isZero()` that allows us to verify if an element is the identity element or not, a method for the computation of the opposite for an element, and a method for the operation $+$. The method `isAddCommutative()` will be defined to return `true` or `false`, depending on the concrete case. If it is possible, this method should be defined static, and implicitly returning `false`.

Rings are other basic algebraic structures, and corresponding to them, we define three interfaces: `RingElem`, `UnitaryRingElem`, `DivisionRingElem`, and `FieldElem` (Figure 1). For fields, which are commutative division rings the corresponding interface is `FieldElem`.

The method `isMultiplyCommutative()` returns `true` or `false` in the ring classes, and always `true` for the field classes.

The method `isInvertible()` returns always `true` for `DivisonRingElem` and `FieldElem` objects. Similarly, the method `isMultiplyCommutative()` always return `true`, in the classes implementing the `FieldElem` interface.

3.2. **Polynomials.** Now we consider the set of all polynomials – $R[X]$, over a commutative ring with identity $(R, +, \cdot)$. $R[X]$ is a subring with identity of the ring $(R^{\mathbb{N}}, +, \cdot)$, where $R^{\mathbb{N}}$ is the set of all functions with the domain $\mathbb{N}$ and codomain R.

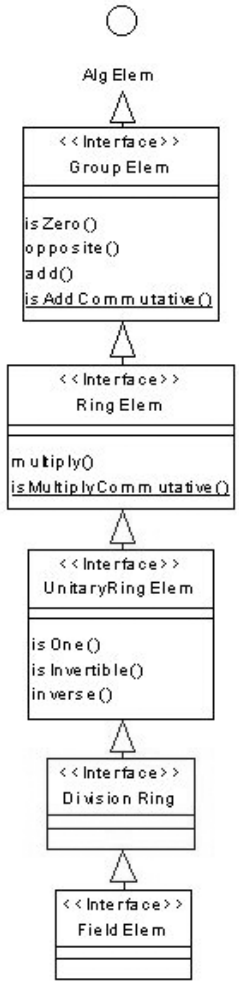


FIGURE 1. The class diagram for the basic structures

Corresponding to these, we consider two interfaces derived from the interface `UnitaryRingElem`: `Polynomial` and `DivisionPolynomial` (Figure 2).

The method `division` has some strong preconditions, assuring that the necessary conditions for polynomials division are satisfied.

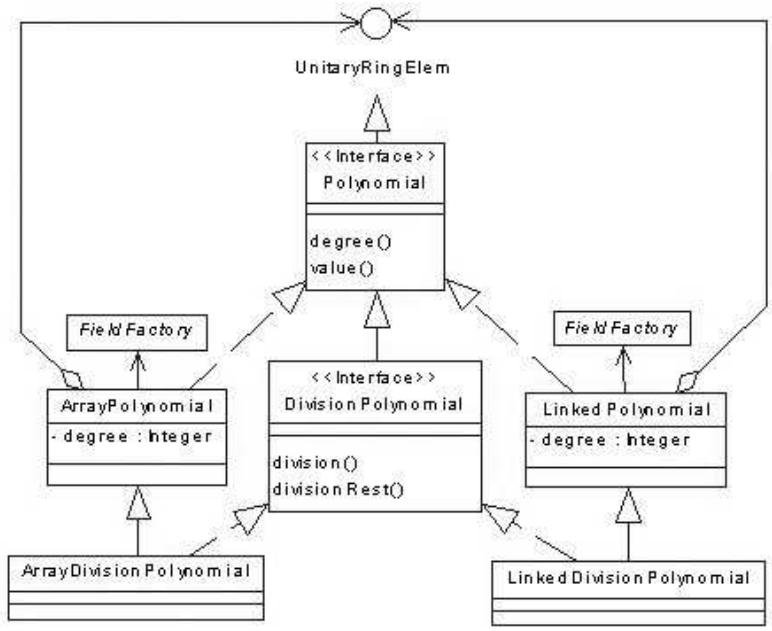


FIGURE 2. The class diagram for polynomials

To implement a concrete class for the ADT Polynomial, we have to choose the representation of the data. There are two classic ways for representation:

- using an array of coefficients;
- using a list of monoms.

We may also consider other storage formats, and, as well, we may introduce *storage format abstraction level* like in [8]. This would mean creating another abstract class `StorageFormat`, from which the specialized storage format classes are derived. Bridge and Iterator patterns have to be used in this case.

In order to better illustrate the associated factory classes, we will consider in this presentation only these two kinds of storage, and we will use simple inheritance.

Using the first representation, the operation of coefficient selection is very fast, because of the direct access. The second one is very useful for the implementation of sparse polynomials.

Corresponding to each of them we define two concrete implementation classes – `ArrayPolynomial`,
`ArrayDivisionPolynomial` and `LinkedPolynomial`, `LinkedDivisionPolynomial`.

The coefficients type is also `UnitaryRingElem`, and so the Composition pattern is used here.

3.3. **Matrices and Vectors.** We start from the definition of the vector space.

Definition 1 (Vector Space)**.** *A vector space over a field K is an abelian group $(V,+)$ together with a so-called external operation*

$$\cdot : K \times V, (k,v) \mapsto k \cdot v,$$

satisfing the following axioms:

(1) $k \cdot (v_1 + v_2) = k \cdot v_1 + k \cdot v_2$;
(2) $(k_1 + k_2) \cdot v = k_1 \cdot v + k_2 \cdot v$;
(3) $(k_1 \cdot k_2) \cdot v = k_1 \cdot (k_2 \cdot v)$;
(4) $1 \cdot v = v$

Theorem 1. *Let V be a vector space over K and $n \in N^*$. Then V^n has a structure of a vector space over K, where the operations are defined by*

$$(v_1, \ldots, v_n) + (v_1^{'}, \ldots, v_n^{'}) = (v_1 + v_1^{'}, \ldots, v_n + v_n^{'}),$$
$$k(v_1, \ldots, v_n) = (kv_1, \ldots, kv_n),$$

where $k \in K$ and $(v_1, \ldots, v_n), (v_1^{'}, \ldots, v_n^{'}) \in V^n$.

For vectors implementation, we will consider the vector space, for which $V = W^n, n \in \mathbb{N}^*$, where $(W,+)$ is an abelian group. Corresponding to it, we build a class `Vector` (Figure 3). The method
`scalarProd(FieldElem)` corresponds to the external operation.

We may consider the matrices to be elements of the vector space $W^{nm}(K) = M_{mn}(K)$. The corresponding class is `SimpleMatrix`.

If the elements of the matrices are unitary ring elements ($(W,+,\cdot)$ forms a unitary ring), we can define a product operation between two matrices A and B that respect the following property: $cols(A) = rows(B) = n$. The product matrix C is defined by:

$$c(i,j) = \sum_{k=0}^{n} a(i,k) \cdot b(k,j).$$

For this case, we defined a class `Matrix` derived from the class `SimpleMatrix` (Figure 3).

Some specific operations can be defined for matrices, such as computation of the rank.

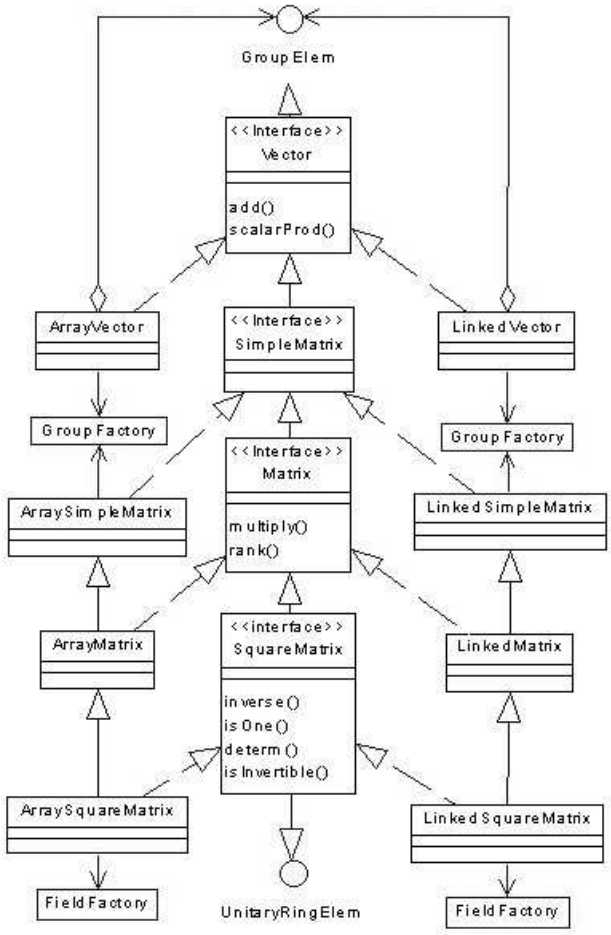


FIGURE 3. The class diagram for Vectors and Matrices

Square matrices with elements from a unitary ring $(R, +, \cdot)$ form also a unitary ring (Figure 3). On square matrices we may introduce many specific operations, such as computation of the determinant, and of the inverse matrix, etc. The class `SquareMatrix` is derived from the class `Matrix`, and it has specific methods.

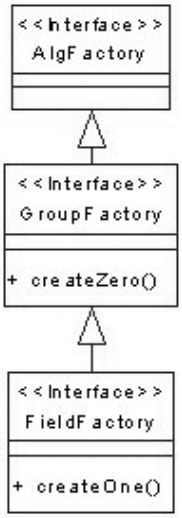


FIGURE 4. The class diagram for the basic factories

For matrices, we may also consider two classic ways for data representation, corresponding to dense and to sparse matrices:

- using a bi-dimensional array of elements;
- using a list of triples.

Corresponding to these representation, we have two kinds of concrete classes for matrices (Figure 3). Like for the polynomials case, Composition design pattern is also used for vectors and matrices.

3.4. **Other Structures and Classes.** The library may be extended with many other structures and classes.

For example, we can add a general abstract class named `AlgebraicSystem`, which allows the user to solve a $n \times n$ algebraic system. The concrete classes derived from it, will implement some concrete solving methods.

3.5. **Factories.** For simetry, we define an empty interface `AlgFactory`, which is the root of factories hierarchy.

The interface `GroupFactory` declare only one method: `createZero():AlgElem`. The interface `FieldFactory` extends the interface GroupFactory, and adds a new

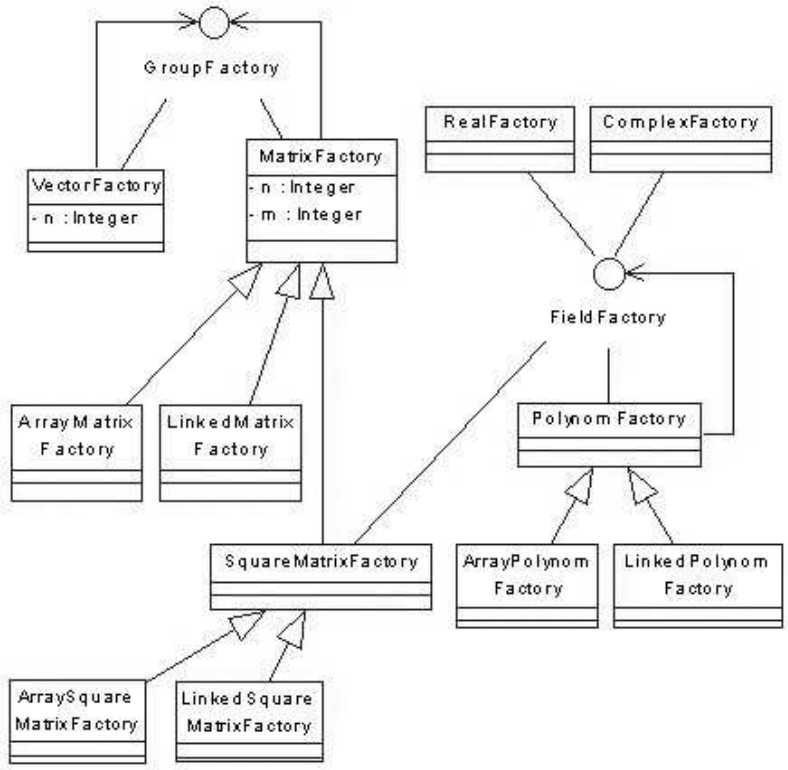


FIGURE 5. The class diagram for some concrete factories

method: `createOne():AlgElem`. We have chosen the name `FieldFactory`, because fields are much more used, but this interface will be also used for the unitary and division rings.

In order to solve the problem of creation for the special value 0 and 1, the implementations of the class `Polynomial` has to use a `FieldFactory` instance. For example, one constructor of the class Polynomial creates an null polynomial. To create a null polynomial, we have to create a null coefficient, and we can do this by using the method `createZero()` of a class that implements `FieldFactory`.

The concrete classes for matrices also have to use a `GroupFactory`. Square matrices classes need for the creation of identity matrices, both methods of a `FieldFactory`.

```
Polynomial p1 = new ArrayPolynomial(RealFactory.getInstance());
Polynomial p2 = new LinkedPolynomial(ComplexFactory.getInstance());
```

FIGURE 6. The creation of null polynomials over $\mathbb{R}$ and over $\mathbb{C}$

```
Matrix m1 = new LinkedMatrix(10,10, RealFactory.getInstance());
Matrix m2 = new ArrayMatrix(10,10, ComplexFactory.getInstance());
```

FIGURE 7. The creation of two 10×10 matrices over $\mathbb{R}$ and over $\mathbb{C}$

```
Matrix m =  new ArraySquareMatrix(10,
            new ArrayPolynomFactory(RealFactory.getInstance()))
```

FIGURE 8. The creation of a null 10×10 matrix over $\mathbb{R}[X]$

`GroupFactory` and `FieldFactory` are basic factories. We may define some other specialized factories, such as: `PolynomFactory, MatrixFactory` (Figure 5). These classes create the null and the unity values for polynomials, and matrices. These factories extend `GroupFactory` and `FieldFactory`, but in the same time they use these interfaces for creation of their basic elements. Their constructors receive a `GroupFactory` or a `FieldFactory` instance, which is used in the functions `createZero()`, or `createOne()`.

For concrete examples, concrete factories have to be built, and examples are given for the real and complex numbers: `RealFactory` and `ComplexFactory`. These concrete factories may implement Singleton[5] pattern.

4. Implementation

Any object oriented language can be chosen.

We have chosen Java because is an almost pure object oriented language, which offers many advantages. We have been very interested in having a good and simple mechanism for exception handling. We need it because there are many preconditions that impose different types of verification – for example, the necessity of checking if a parameter instance has a specific type. Our implementation in Java, define a main package named `Algebra`, which has some subpackages: `Factories, BasicStructures`, and `Structures`.

5. Concrete Examples

Let's consider that we need to work with polynomials with coefficients' types: real and complex. First, we define two classes `Real` and `Complex`, which implement `FieldElem` interface. Also, two factory classes has to be built: `RealFactory` and `ComplexFactory`, which implement the interface `FieldFactory`. The code showed in the Figure 6 illustrates the creation of two null polynomials: one over the $\mathbb{R}$ and another over the $\mathbb{C}$.

Similarly, we can work with matrices with elements of type real and complex, without creating any other class (Figure 7).

But, we can also work with matrices that have polynomial elements, like it is showed in the Figure 8.

6. Conclusions and Future Work

We have defined here a design scheme for a general algebraic library. The design scheme is based on object oriented programming, and it offers generality and flexibility. The Abstract Factory and Singleton creational design patterns, and some other design patterns, such as Composition, have been used.

The library is designed in a way that offers the user the possibility of working with general algebraic structures without concerning about the types dependencies.

Some general abstract algebraic structures and also some basic concrete algebraic structures have been designed. The users may define, without any problems, other algebraic structures based on the abstract ones. Also, existing algebraic structures may be combined and used in different ways. These are possible mainly because of using creational design patterns.

The fact that OOP allows us to describe problems in the terms of the problem space, rather than the terms of the solution space[3], is very well emphasized here. We can work and reasoning with mathematical abstractions as: groups, fields, rings, ...

This design scheme represents the first step for developing a general algebraic library; in the next step we intend to introduce the storage format abstraction level, which is based on Bridge and Iterator patterns. We intend to combine the storage format abstraction level with Abstract Factory pattern, in a way in which the factory will be also responsible for the creation of the specialized storage.

References

[1] Arnold, K., Holmes, D., Gosling, J., *The Java Programming Language*, Addison-Wesley, 2000.

[2] BLAS Technical Forum, Document for the BAsic Linear Algebra Subprograms, http://www.netlib.org/blas/blast-forum/

[3] Eckel, B., *Thinking in Java*, http://www.EckelObjects.com, 2000.
[4] Eriksson, H.E., Penker, M. *UML Toolkit*, Wiley Computer Publishing, 1997.
[5] Gamma, E., Helm, R., Johnson, R., Vlissides, J. *Design Patterns: Elements of Reusable Object Oriented Software*, Addison-Wesley, 1995.
[6] Gilbert, J., *Elements of Modern Algebra*, PWS-Kent, Boston, 1992.
[7] LAPACK++: A Design Overview of Object Oriented Extensions for High Performance Linear Algebra, In Proceeedings of SuperComputing'93, pg. 162-171, IEEE Computer Society Press, 1993.
[8] Luján, M., Freeman, T. L., Gurd, J. R., *OoLaLa: an Object Oriented Analysis and Design of Numerical Linear Algebra*, In the Proceedings of Conference on Object-Oriented Programming, Systems, Languages, and Applications – OOPSLA 2000.
[9] Musser, D.R., Scine A., *STL Tutorial and Reference Guide: C++ Programming with Standard Template Library*, Addison-Wesley, 1995.
[10] Myers A.C., Bank J.A., Liskov Barbara, *Parameterized Types for Java*, Proceadings of the 24th ACM Symposium on the Principles of Programming Languages, Paris, 1997.

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

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLVIII**, Number 1, 2003

SRDJAN STOJANOVIC, "COMPUTATIONAL FINANCIAL MATHEMATICS USING MATHEMATICA: OPTIMAL TRADING STOCKS AND OPTIONS", BIRKHÄUSER VERLAG, BOSTON-BASEL-BERLIN, 2003, XI+481 PAGES

DIANA ANDRADA FILIP

The book consists in 481 pages i.e. 8 chapters, a bibliography and an index and includes CD-ROM. Srdjan Stojanovic taught the course on Financial Mathematics at the University of Cincinnati since 1998 and at Purdue University during the academic year 2001-2002. This book is an expanded version of those courses, built with the help of the students during the time when Srdjan Stojanovic taught them computational financial mathematics and MATHEMATICA^R programming.

A very interesting and very actual book, because now, the computer make an integrand part of our life. The author, himself, underlines in the Introduction, that the book is addressed to students and professors of academic programs in financial mathematics (like computational finance and financial engineering). Anyway, the mathematical background would be Calculus, Differential Equations and Probability, but varies according to the objectives of the reader. The book is, as recommends the author, divided in some parts according to the required mathematical level as follows: the basics (for the Chapters 1-4), intermediate level (the Chapters 5 and 7), advanced level (for the Chapters 6 and 8).

In the Chapter 1, **Cash Account Evolution**, ordinary differential equations are solving with Mathematica^R, and symbolic and numerical solutions of ODEs are presented.

The Chapter 2, **Stock Price Evolution**, explains to the reader what are stocks and then presents the stock price modeling, i.e. some stochastic differential equations. An other aim of this chapter is to be acquainted with Itô calculus and with multivariable and symbolic Itô calculus. Also, some relationship between SDEs and PDEs are presented.

In the Chapter 3, **European Style Stock Options**, the first paragraph deals with the notion of stock option. Then, the Black and Scholes PDE and hedging are presented and the Black and Scholes PDE are symbolically solved. Also, the generalized Black and Scholes formulas are presented.

In the Chapter 4, **Stock Market Statistics**, the stock market data import and manipulation are presented. Then, the chapter deals with the volatility estimates,

Received by the editors: April 15, 2003.

i.e. scalar case, and also deals with the appreciation rate estimates (the scalar case) and the statistical experiments (Bayesian and non-Bayesian). In the same chapter, the vector basic price model statistics and the dynamic statistics, like the filtering of conditional Gaussian processes, are treated.

In the Chapter 5, **Implied Volatility for European Options**, the option market data is presented. After that, the Black and Scholes theory is made obvious vs. market data (the implied volatility) and then, the numerical PDEs, the optimal control and the implied volatility are studied.

The Chapter 6, **American Style Stock Options**, deals with the american options, the obstacle problems and presents the general implied volatility for american options.

Very important, the Chapter 7, **Optimal Portfolio Rules,** presents the utility of wealth, the Merton's optimal portfolio rule derived and implemented, the portfolio rules under appreciation rate uncertainty, the portfolio optimization under equality constraints, the portfolio optimization under inequality constraints.

In the Chapter 8, **Advanced Trading Strategies,** the reduced Monge-Ampère PDEs of advanced portfolio hedging and the hypoelliptic obstacle problems in optimal momentum trading are presented.

As we have already said, the book is accompanied by a CD-ROM, but the book is not a software product. Informations about further developments might be available at the web site CFMLab.com. The reader may direct comments to the same address.

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

E-mail address: dafilip@math.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLVIII**, Number 1, 2003

YORICK HARDY AND WILLI-HANS STEEB, "CLASSICAL AND QUANTUM COMPUTING WITH C++ AND JAVA SIMULATIONS", BIRKHÄUSER VERLAG, BASEL-BOSTON-BERLIN, 2001

DAN DUMITRESCU

Scientific computing is not numerical analysis, the analysis of algorithms, high performance computing or computer graphics. It consists instead of the combination of all these fields and others to craft solution strategies for applied problems. It is the original application area of computers and remains the most important. From meteorology to plasma physics, environmental protection, nuclear energy, genetic engineering, symbolic computation, network optimization, financial applications and many other fields, scientific applications are larger, more ambitious, more complex, and more necessary. More and more universities introduce a Department of Scientific Computing or a Department of Computational Science. The components of this new department include Applied Mathematics, Theoretical Physics, Computer Science and Electronic Engineering.

Classical and Quantum Computing provides a self-contained, systematic and comprehensive introduction to all the subjects and techniques important in scientific computing. The style and presentation are readily accessible to undergraduates and graduates. A large number of examples, accompanied by complete C++ and Java code wherever possible, cover every topic.

Features and benefits:

- comprehensive coverage of the theory with many examples;
- website with programs and support is given at "http://issc.rau.ac.za" "http://issc.rau.ac.za";
- topics in classical computing include Boolean algebra, gates, circuits, latches, error detection and correction, neural networks, Turing machines, cryptography, genetic algorithms;
- for the first time, genetic expression programming is presented in a textbook topics in quantum computing include mathematical foundations, quantum algorithms, quantum information theory;
- hardware used in quantum computing.

Received by the editors: April 15, 2003.

This book serves as a textbook for courses in scientific computing and is also very suitable for self-study. Students, professionals and practitioners in computer science, applied mathematics and physics will benefit from using the book and the included software simulations.

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

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

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLVIII**, Number 1, 2003

BRIGITTE CHAUVIN, PHILIPPE FLAJOLET, DANIÈLE GARDY, ABDELKADER MOKKADEM EDS., "MATHEMATICS AND COMPUTER SCIENCE II: ALGORITHMS, TREES, COMBINATORICS AND PROBABILITIES", BIRKHÄUSER VERLAG, BASEL-BOSTON-BERLIN, 2002, ISBN 3-7643-6933-7, 557 PAGES

HORIA F. POP

This book features a collection of original papers situated at the cross-roads of Mathematics and Computer Science, representing the Proceedings of the International Colloquium of Mathematics and Computer Science, held at the University of Versailles-St-Quentin, in September 18–20, 2002. The issue is centered around topics of large interest, as Combinatorics, Random Graphs and Networks, Algorithms Analysis and Trees, Branching Processes and Trees, Applied Random Combinatorics. The book has 557 pages, and provides 34 papers written by 60 authors, distributed among five distinct chapters of mathematics and computer science.

Combinatorics is the starting point of many researches of discrete models. A few important results are presented, concerning map colouring problems, a theory of walks, ECO-systems, planar maps.

Ramdon Graphcs and Networks have been the subject of intense study for fourty years. A few results concern an analysis of triangle-free graphs on breadth-first search, random maps and random graphs, colouring random graphs, aproximability of paths colouring problems, monimal spanning trees.

Analysis of Algorithms and Trees. Trees appear as data structures in a variety of domains, like data processing, data compression, information retrieval, symbolic computation. Among the published studies, we remind the analysis of the new suffix search tree data structure, the analysis of the Quickfind algorithm, a study of digit statistics in a variety of number representation systems.

Brancing Processes and Trees. Branching processes are the probabilistic counterparts of the combinatorial theory of trees. The published studies include analyses of random trees, random walks, stable weighted branching processes.

Applied random combinatorics. A few papers present implications of random combinatorics to many other areas of science: the parking problem, sensitivity

Received by the editors: May 15, 2003.

of Boolean functions to input noise, new results in learning theory, nierarchically structured databases, and a study of the number of spanning tres in structured graphs.

This book illustrates numerous ramifications of the theory of random discrete structures throughout mathematics and computer science. The book serves both as a reference text and as a smooth introduction to many aspects of interest both to mathematicians and computer scientists. It is an outstanding tool and a main information source for a large public in applied mathematics, discrete mathematics and computer science, including researhcers, teachers, graduate students adn engineers.

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

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