STUDIA UNIVERSITATIS
BABEŞ-BOLYAI

INFORMATICA

1/2008

Anul LIII 2008

STUDIA

UNIVERSITATIS BABEŞ-BOLYAI

INFORMATICA

1

Redacţia: 400084 Cluj-Napoca, Str. M. Kogălniceanu nr. 1 Tel: 405300

SUMAR □ CONTENTS □ SOMMAIRE

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LIII**, Number 1, 2008

A SEARCH BASED APPROACH FOR IDENTIFYING DESIGN PATTERNS

GABRIELA ŞERBAN AND ISTVÁN GERGELY CZIBULA

Abstract. *Software design patterns* are well-known and frequently reused micro-architectures: they provide proved solutions to design recurring problems with certain contexts. In restructuring legacy code it is useful to introduce a design pattern in order to add clarity to the system and thus facilitate further program evolution. That is why the problem of design patterns identification is very important. Automating the detection of design pattern instances could be of significant help to the process of reverse engineering large software systems. The aim of this paper is to introduce a new search based approach for identifying instances of design patterns in a software system. We provide an experimental evaluation of our approach, emphasizing its advantages.

1. Introduction

Design patterns have attracted significant attention in software engineering in the last period. An important reason behind this is that design patterns are potentially useful in both development of new, and comprehension of existing object-oriented design, especially for large legacy systems without sufficient documentation. The design patterns introduced by Gamma et al. [3] capture solutions that have developed and evolved over time. Each design pattern indicates a high level abstraction, encompasses expert design knowledge, and represents a solution to a common design problem. A pattern can be reused as a building block for better software construction and designer communication.

In restructuring legacy code is useful to introduce a design pattern in order to add clarity to the system and thus facilitate further program evolution. That is why the problem of design patterns identification is very important. Automating the detection of design pattern instances could be of significant help to the process of reverse engineering large software systems.

Received by the editors: December 4, 2007.

2000 *Mathematics Subject Classification.* 68N99, 62H30.

1998 *CR Categories and Descriptors.* D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement –*Restructuring, reverse engineering, and reengineering*; I.5.3 [**Computing Methodologies**]: Pattern Recognition – *Clustering*.

From a program understanding and reverse engineering perspective, extracting information from a design or source code is very important, the complexity of this operation being essential. Localizing instances of design patterns in existing software can improve the maintainability of software. Automatic detection of design pattern instances is probably a useful aid for maintenance purposes, for quickly finding places where extensions and changes are most easily applied.

It would be useful to find instances of design patterns especially in designs where they were not used explicitly or where their use is not documented. This could improve the maintainability of software, because larger chunks could be understood as a whole.

The presence of patterns in a design should be reflected also in the corresponding code: being able to extract pattern information from both design and code is fundamental in identifying traceability links between different documents, explaining the rationale of the chosen solution in a given system and thus simplifying the activity of building its conceptual model.

The main contributions of this paper are:

- To introduce an original search based approach for identifying instances of design patterns in a software system.
- To emphasize the advantages of the proposed approach in comparison with existing approaches.

The rest of the paper is structured as follows. Section 2 presents some existing approaches in the field of automatic design patterns identification. The theoretical model of our search based approach for identifying design patterns is introduced in section 3. Section 4 presents our approach and Section 5 contains an experimental evaluation of it. An analysis of the proposed approach is made in Section 6. Section 7 presents some conclusions of the paper and future research directions.

2. Related Work

We will briefly present, in the following, the most significant results obtained in the literature in the field of automatic design patterns identification.

Different approaches, exploiting software metrics, were used in previous works to automatically detect design concepts and function clones [6, 7] in large software systems. An approach for extracting design information directly from C++ header files and for storing them in a repository is proposed in [7]. The patterns are expressed as PROLOG rules and the design information is translated into facts. A single Prolog query is then used to search for all patterns. The disadvantage of this approach is that handles a small number of design patterns (only the structural design patterns – Adapter, Bridge, Composite, Decorator and Proxy) and the precision obtained in recognition is small (40%).

A multi-stage approach using OO software metrics and structural properties to extract structural design patterns from object oriented artifacts, design, or code, is introduced in [1]. The drawback of this approach is that only few pattern families (the structural design patterns - Adapter, Bridge, Composite, Decorator and Proxy) are considered.

In [8] the authors have developed an iterative semiautomatic approach to design recovery using static analysis on the source code level of a system. The approach facilitates a rule-based recognition of design pattern instances. It is a highly scalable process which can be applied to large real world applications with more than 100,000 LOC. The approach has been enhanced by fuzzyfied rules to provide the reverse engineer with accuracy information about the analysis results [9]. Fuzzyfied rules have a credibility value expressing how much design pattern candidates identified by the rule are real design pattern instances. The reverse engineer adapts the accuracy values of the results which are then used to calibrate credibility values of the rules [4].

For a precise design pattern recognition, a static analysis is not sufficient. The behavioral aspects of a pattern are an important factor. Dynamic analyses can be used to analyze the runtime behavior of a system. A sole dynamic analysis is not feasible since the amount of data gathered during runtime is too big. A combination of static and dynamic analysis techniques is proposed in [13]. The static analysis identifies candidates for design pattern instances. These candidates form a significantly reduced search space for a subsequent dynamic analysis that confirms or weakens the results from static analysis.

3. Theoretical model

Let $S = \{s_1, s_2, ..., s_n\}$ be a software system, where $s_i, 1 \leq i \leq n$ may be an application class, a class method or a class attribute. We will refer an element of the software system S as an *entity*.

Let us consider that:

- $Class(S) = \{C_1, C_2, \ldots, C_l\}$, $Class(S) \subset S$, is the set of applications classes in the initial structure of the software system S.
- Each application class C_i $(1 \leq i \leq l)$ is a set of methods and attributes, i.e., $C_i = \{m_{i1}, m_{i2}, \ldots, m_{ip_i}, a_{i1}, a_{i2}, \ldots, a_{ir_i}\},\ 1 \leq p_i \leq n,\ 1 \leq r_i \leq n$, where m_{ij} $(\forall j,\ 1 \leq j \leq p_i)$ are methods and a_{ik} $(\forall k,\ 1 \leq k \leq r_i)$ are attributes from C_i.
- $Meth(S) = \bigcup\limits_{i=1}^{l} \bigcup\limits_{j=1}^{p_i} m_{ij}$, $Meth(S) \subset S$, is the set of methods from all the application classes of the software system S.

- $Attr(S) = \bigcup\limits_{i=1}^{l} \bigcup\limits_{j=1}^{r_i} a_{ij}$, $Attr(S) \subset S$, is the set of attributes from the application classes of the software system S.

Based on the above notations, the software system S is defined as follows:

$$S = Class(S) \bigcup Meth(S) \bigcup Attr(S). \tag{1}$$

A given *design pattern* p from the software system S can be viewed as a pair $p = (\mathcal{C}_p, \mathcal{R}_p)$, where

- $\mathcal{C}_p = \{C_1^p, C_2^p, \ldots, C_{nc_p}^p\}$ is a subset from $Class(S)$, $\mathcal{C}_p \subset Class(S)$, and represents the set of classes that are components of the design pattern p. nc_p represents the number of classes from the pattern p.
- $\mathcal{R}_p = \{r_1^p, r_2^p, \ldots, r_{nr_p}^p\}$ is a set of constraints (relations) existing among the classes from $\mathcal{C}_p$, constraints that characterize the design pattern p. Consequently, each constraint $r_i^p, \forall 1 \leq i \leq nr_p$ from $\mathcal{R}_p$ is a relation defined on a subset of classes from $\mathcal{C}_p$, and nr_p represents the number of constraints characterizing the design pattern p.

We mention that all the constraints from $\mathcal{R}_p$ can be expressed as binary constraints (there are two classes involved in the constraint). That is why, in the following, we will assume, without loosing generality, that all the constraints from $\mathcal{R}_p$ are binary.

Let us denote by $cmin$ the minimum number of binary constraints from $\mathcal{R}_p$ that a class from $\mathcal{C}_p$ can satisfy, as indicated by equality (2).

$$cmin = min_{i=1,nc_p} |\{j,\ |1 \leq j \leq nr_p,\ \exists 1 \leq k \leq nc_p, k \neq i\ s.t.\ C_i^p\ r_j^p\ C_k^p \vee C_k^p\ r_j^p\ C_i^p\}| \tag{2}$$

4. Our search-based approach

In this section we are focusing on identifying all the instances of a given design pattern p in a given design (software system).

Based on the theoretical model defined in Section 3, it can be easily seen that the problem of identifying all instances of the design pattern p in the software system S is a *constraint satisfaction problem* [10], i.e., the problem of searching for all possible combinations of nc_p classes from S such that all the constraints from $\mathcal{R}_p$ to be satisfied.

It is obvious that a brute force approach for solving this problem would lead to a worst case time complexity of $O(l^{nc_p})$. The main goal of the search based approach that we propose in this section in order to find all instances of a design pattern p is to reduce the time complexity of the process of solving the analyzed problem.

The main idea of our approach is to obtain a set of possible pattern candidates (by applying a preprocessing step on the set $Class(S)$) and then to apply a hierarchical clustering algorithm in order to obtain all instances of the design pattern p.

Our search-based approach for identifying instances of design patterns in a software system consists of the following steps:

- **Data collection**: The existing software system is analyzed in order to extract from it the relevant entities: classes, methods, attributes and the existing relationships between them.
- **Preprocessing**: From the set of all classes from S we eliminate all the classes that can not be part of an instance of pattern p. This preprocessing step will be explained later.
- **Grouping**: The set of classes obtained after the **Preprocessing** step are grouped in clusters using a hierarchical clustering algorithm. The aim is to obtain clusters with the instances of p (each cluster containing an instance) and clusters containing classes that do not represent instances of p.
- **Design pattern instances recovery**: The clusters obtained at the previous step are filtered in order to obtain only the clusters that represent instances of the design pattern p.

In the following we will give a descriptions of the above enumerated steps.

4.1. **Data collection.** During this step, the existing software system is analyzed in order to extract from it the relevant entities: classes, methods, attributes and the existing relationships between them. In order to verify the constraints $\mathcal{R}_p$ of the design pattern p, we need to collect from the system information such as: all interfaces implemented by a class, the base class of each class, all methods invoked by a class, all possible concrete types for a formal parameter of a method, etc.

In order to express the dissimilarity degree between any two classes relating to the considered design pattern p, we will consider the distance $d(C_i, C_j)$ between two classes C_i and C_j from S given by the number of binary constraints from $\mathcal{R}_p$ that are not satisfied by classes C_i and C_j. It is obvious that as smaller the distance d between two classes is, as it is more likely that the two classes are in an instance of the design pattern p. The distance is expressed as in formula (3).

(3)

$$d(C_i, C_j) = \begin{cases} 1 + |\{k \mid 1 \leq k \leq nr_p \ s.t. \ \neg(C_i \ r_k^p \ C_j \vee C_j \ r_k^p \ C_i)\}| & i \neq j \\ 0 & i = j \end{cases}.$$

Based on the definition of d given above it can be simply proved that d is a semimetric function.

4.2. **Preprocessing.** After the **Data collection** step was performed and the needed data was collected from the software system in order to compute the *distances* between the classes (3), a preprocessing step is performed in order to reduce the search space, i.e, the set of possible pattern candidates.

In order to significantly reduce the search space, we eliminate from the set of all classes $Class(S)$ those classes that certainly can not be part of an instance of the design pattern p. By applying this filtering, we will obtain a set of possible pattern candidates, denoted by $PatCand(S)$. More specifically, the following filtering step is performed:

- We eliminate from the set of all classes those classes that satisfy less than $cmin$ binary constraints from $\mathcal{R}_p$. The idea is that based on the definition of $cmin$ given by (2), in order to be in an instance of design pattern p, a class has to satisfy at least $cmin$ constraints from $\mathcal{R}_p$. After the filtering step, the set of pattern candidates becomes:

$$PatCand(S) = Class(S) - \{C_j | 1 \leq j \leq l \ s.t. \sum_{i=1, i \neq j}^{l} (1 + nr_p - d(C_j, C_i)) < cmin\}$$

After applying the previous filtering step, the set $PatCand(S)$ of possible pattern candidates is significantly reduced in comparison with the set of all classes from S. Let us denote by nc the number of possible pattern candidates, i.e, the cardinality of the set $PatCand(S)$.

4.3. **Grouping.** After the grouping step we aim to obtain a partition $\mathcal{K} = \{K_1, K_2, ..., K_v\}$ of the set $PatCand(S)$ such that each instance of the design pattern p to form a cluster. Based only on the distance semimetric d (3), it is possible that two classes would seem to be in an instance of the design pattern (a so called "false positive" decision), even if they are not cohesive enough in order to take this decision. That is why we need a measure in order to decide how cohesive are two classes.

We will adapt the generic cohesion measure introduced in [12] that is connected with the theory of similarity and dissimilarity. In our view, this cohesion measure is the most appropriate to our goal. We will consider the dissimilarity degree (from the cohesion point of view) between any two classes from the software system S. Consequently, we will consider the dissimilarity $diss(C_i, C_j)$ between classes C_i and C_j as expressed in (4).

$$(4) \qquad diss(C_i, C_j) = \begin{cases} 1 - \frac{|p(C_i) \cap p(C_j)|}{|p(C_i) \cup p(C_j)|} & if \;\; p(C_i) \cap p(C_j) \neq \emptyset \\ \infty & otherwise \end{cases},$$

where $p(C)$ defines a set of relevant properties of class C and it consists of the application class itself, all attributes and methods defined in the class C, all interfaces implemented by C and the base class of C.

We have chosen the dissimilarity between two classes as expressed in (4) because it emphasizes the idea of cohesion. As illustrated in [2], "*Cohesion refers to the degree to which module components belong together*". The dissimilarity measure defined in Equation (4) highlights the concept of cohesion, i.e., classes with low dissimilarities are cohesive, whereas classes with higher distances are less cohesive.

Based on the definition of *diss* (4), it can be easily proved that *diss* is a semimetric function.

In the original paper [11] a theoretical validation of the *semimetric* dissimilarity function *diss* is given. It is proved that *diss* highlights the concept of cohesion, i.e., classes with low distances are cohesive, whereas classes with higher distances are less cohesive.

Consequently, the dissimilarity semimetric *diss* can be used in order to decide how cohesive are two classes. We will use *diss* in the **Grouping** step of our approach in order to decide if two classes are cohesive enough in order to be part of an instance of the design pattern.

In order to obtain the desired partition $\mathcal{K}$, we introduce a *hierarchical agglomerative clustering algorithm* (HAC).

In our approach the objects to be clustered are the classes from the set $PatCand(S)$ and the distance function between the objects is given by the semimetric d (3). We use *complete-link* [5] as linkage metric between the clusters, because it is the most appropriate linkage metric to our goal. Consequently, the distance $dist(k, k')$ between two clusters $k \in \mathcal{K}$ and $k' \in \mathcal{K}$ ($k \neq k'$) is given as in (5).

$$(5) \qquad dist(k, k') = max_{e \in k, e' \in k'} d(e, e')$$

In the hierarchical clustering process, the dissimilarity semimetric *diss* will be used in order to decide how cohesive are two classes, i.e, if they will be merged or not in the same cluster. That is why we will consider the *dissimilarity* between two clusters $k \in \mathcal{K}$ and $k' \in \mathcal{K}$ (denoted by $dissimilarity(k, k')$) as the maximum dissimilarity between the objects from the clusters:

$$dissimilarity(k,k') = max_{e \in k, e' \in k'} diss(e,e') \tag{6}$$

The main steps of HAC algorithm are:

- Each class from $PatCand(S)$ is put in its own cluster (singleton).
- The following steps are repeated until the partition of classes remains unchanged (no more clusters can be selected for merging):
  - Select the two most similar clusters from the current partition, i.e, the pair of clusters that minimize the distance from (5). If this selection is nondeterministic (there are several pair of clusters with the same minimum distance between them), we will choose the pair (K_i, K_j) that has the minimum associated dissimilarity value ($dissimilarity(K_i, K_j)$). Let us denote by $dmin$ the distance between the most similar clusters K_i and K_j.
  - If $dmin < 1 + nr_p$ (nr_p is the number of constraints imposed by the design pattern p), then clusters K_i and K_j will be merged, otherwise the partition remains unchanged. The idea of this step is that two clusters will not be merged if their most distant classes can not be part of an instance of the design pattern p (they invalidate all the constraints that must hold).

We give next HAC algorithm.

```
Algorithm HAC is
Input: - the set of possible pattern candidates $PatCand(S)$,
        - the semimetric $d$,
        - the semimetric $diss$,
        - the number $nr_p$ of imposed constraints.
Precondition: - $l \geq 2$.
Output: - the partition $\mathcal{K} = \{K_1, K_2, ..., K_v\}$.
Begin
    $v \leftarrow |PatCand(S)|$ //the number of possible pattern candidates
    For each $C \in PatCand(S)$ do
        $K_i \leftarrow \{C\}$ //each possible candidate is put in its own cluster
    endfor
    $\mathcal{K} \leftarrow \{K_1, \ldots, K_v\}$ //the initial partition
   $change \leftarrow$ true
   While $change$ do //while $\mathcal{K}$ changes
       //the most similar clusters are chosen for merging
       $dmin \leftarrow \infty$ //the minimum distance between clusters
       $dissmin \leftarrow \infty$ //the minimum dissimilarity between clusters
       For $i^* \leftarrow$ 1 to $v$-1 do //the most similar clusters are chosen
```

```
      For $j^* \leftarrow i^* + 1$ to $v$ do
        $d \leftarrow dist(K_{i^*}, K_{j^*})$ //the distance between the clusters
        If $d < dmin$ then
          $dmin \leftarrow d$
          $i \leftarrow i^*$
          $j \leftarrow j^*$
        else
          If $d = dmin$ then
            $dss \leftarrow dissimilarity(K_{i^*}, K_{j^*})$ //the dissimilarity between the clusters
            If $dss < dissmin$ then
              $dissmin \leftarrow dss$
              $i \leftarrow i^*$
              $j \leftarrow j^*$
            endif
          endif
        endif
      endfor
    endfor
    If $dmin < 1 + nr_p$ then
      $K_{new} \leftarrow K_i \cup K_j$
      $\mathcal{K} \leftarrow (\mathcal{K} \setminus \{K_i, K_j\}) \cup \{K_{new}\}$
      $v \leftarrow v - 1$
    else
      $change \leftarrow$ false //the partition remains unchanged
    endif
  endwhile
  //$\mathcal{K} = \{K_1, K_2, ..., K_v\}$ is the output partition
End.
```

4.4. **Design pattern instances recovery.** The partition $\mathcal{K}$ obtained after the **Grouping** step will be filtered in order to obtain only the clusters that represent instances of the design pattern p. A cluster k from the partition $\mathcal{K}$ is consider an instance of design pattern p iff the classes from k verify all the constraints from $\mathcal{R}_p$ (the set of constraints imposed by the design pattern p).

5. Experimental Evaluation

In our experiment, we are focusing on identifying instances of *Proxy* design pattern using the search based approach that we have introduced in the previous section.

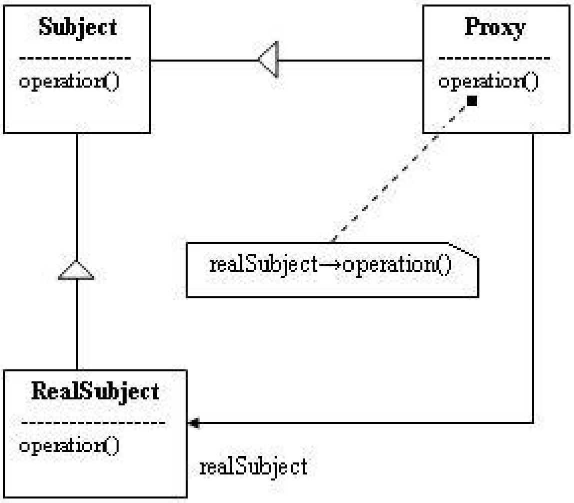


Figure 1. *Proxy* design pattern.

5.1. **The *Proxy* design pattern.** The class diagram of the *Proxy* [3] design pattern is given in Figure 1.

A Proxy pattern constitutes use of proxy objects during object interaction. A proxy object acts as a substitute for the actual object Provide a surrogate or placeholder for another object to control access to it

Proxy is a structural design pattern that provides a surrogate or placeholder for another object to control access to it. Use of proxy objects is prevalent in remote object interaction protocols (*Remote proxy*): a local object needs to communicate with a remote process but we want to hide the details about the remote process location or the communication protocol. The *proxy* object allows to access remote services with the same interface of local processes. In fact, when an *Operation* is required to the proxy object, it delegates the implementation of the required operation to the *RealSubject* object. Being both *Proxy* and *RealSubject* subclasses of *Subject*, this guarantees that they export the same interface for *Operation*. To be able to call *RealSubject* methods, *Proxy* needs an association to it.

According to the considerations from Section 3, the design pattern *proxy* can be defined as the pair $proxy = (\mathcal{C}_{proxy}, \mathcal{R}_{proxy})$, where:

- $\mathcal{C}_{proxy} = \{C_1, C_2, C_3\}$, and $nc_{proxy} = 3$ (the number of classes involved in the design pattern *proxy* is 3).
- $\mathcal{R}_{proxy} = \{r_1, r_2, r_3\}$, $nr_{proxy} = 3$ (the number of constraints imposed by the design pattern *proxy* is 3), and the constraints are:

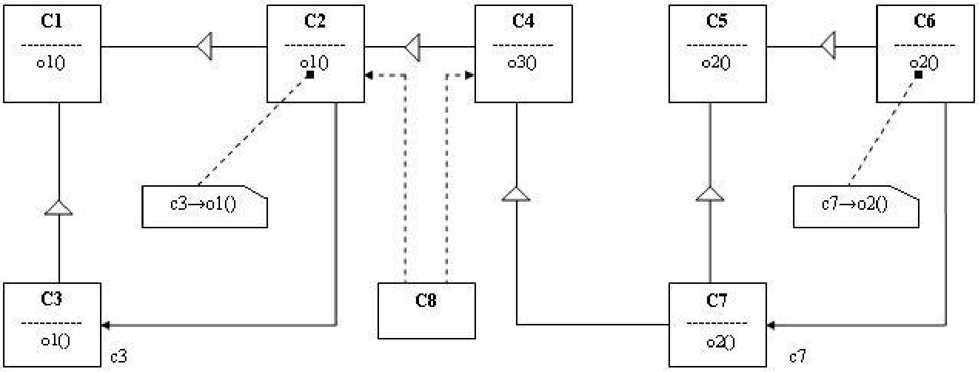


FIGURE 2. The example design S.

- $r_1(C_1, C_2)$ represents the relation "C_2 extends C_1".
- $r_1(C_1, C_3)$ represents the relation "C_3 extends C_1".
- $r_1(C_2, C_3)$ represents the relation "C_2 delegates any method inherited from a class C to C_3, where both C_2 and C_3 extend C".

Considering the above, the minimum number of binary constraints $cmin$ from $\mathcal{R}_p$ that a class from $\mathcal{C}_p$ can satisfy (as indicated in (2)) is 2.

5.2. **Example.** Let us consider as a case study the simple design illustrated in Figure 2.

For the analyzed design S, the set of classes is $Class(S) = \{C_1, C_2, C_3, C_4, C_5, C_6, C_7, C_8\}$ and the number of classes is $l = 8$.

After performing the **Data collection** step from our approach, the matrix $D(8, 8)$ (where a line i corresponds to class C_i and a column j corresponds to class C_j) of distances between the classes from $Class(S)$ is:

$$D = \begin{pmatrix} 0 & 3 & 3 & 4 & 4 & 4 & 4 & 4 \\ 3 & 0 & 3 & 3 & 4 & 4 & 4 & 4 \\ 3 & 3 & 0 & 4 & 4 & 4 & 4 & 4 \\ 4 & 3 & 4 & 0 & 4 & 4 & 3 & 4 \\ 4 & 4 & 4 & 4 & 0 & 3 & 3 & 4 \\ 4 & 4 & 4 & 4 & 3 & 0 & 3 & 4 \\ 4 & 4 & 4 & 3 & 3 & 3 & 0 & 4 \\ 4 & 4 & 4 & 4 & 4 & 4 & 4 & 0 \end{pmatrix} \tag{7}$$

After the **Preprocessing** step, the set of possible pattern candidates is computed, $PatCand(S) = \{C_1, C_2, C_3, C_4, C_5, C_6, C_7\}$ and the number of possible pattern candidates is $nc = 7$. At this step, class C_8 is eliminated because it does not satisfy any constraint from the set of all constraints imposed by the *Proxy* design pattern.

Now the **Grouping** step will be performed and first the matrix $DISS(7, 7)$ (where a line i corresponds to class C_i and a column j corresponds to class C_j) of dissimilarities between the classes from $PatCand(S)$ will be computed:

$$DISS = \begin{pmatrix} 0 & 0.87 & 0.83 & \infty & \infty & \infty & \infty \\ 0.87 & 0 & 0.75 & 0.88 & \infty & \infty & \infty \\ 0.83 & 0.75 & 0 & \infty & \infty & \infty & \infty \\ \infty & 0.88 & \infty & 0 & \infty & \infty & 0.87 \\ \infty & \infty & \infty & \infty & 0 & 0.87 & 0.85 \\ \infty & \infty & \infty & \infty & 0.87 & 0 & 0.77 \\ \infty & \infty & \infty & 0.87 & 0.85 & 0.77 & 0 \end{pmatrix} \tag{8}$$

After applying HAC clustering algorithm, the obtained partition of $PatCand(S)$ is $\mathcal{K} = \{K_1, K_2, K_3\}$, where $K_1 = \{C_4\}$, $K_2 = \{C_3, C_1, C_2\}$ and $K_3 = \{C_5, C_6, C_7\}$.

We mention that without using the dissimilarity matrix $DISS$, the class C_7 would have been grouped with the class C_4, instead of being grouped with classes C_5 and C_6 and an instance of the design pattern *Proxy* would have been missed.

Now we analyze the obtained partition $\mathcal{K}$ in order to identify instances of *Proxy* design pattern, and the identified instances are correctly reported: K_2 and K_3.

6. Analysis of our approach

In the following we will make a time complexity analysis of our search based approach for identifying instances of design patterns in a given software system S. Let us consider that n is the number of entities from S and l is the application classes from S.

Usually, the number nr_p of constraints in a design pattern p is a small constant (as 3 for the *Proxy* design pattern), that is why we will ignore it in the worst time complexity asymptotic analysis.

Analyzing the steps performed in order to identify the instances of design patterns in a given software system (as indicated in Section 4) we can compute their worst time complexity. The results are given in Table 1.

Step	**Worst time complexity**
Data collection	$O(n)$
Preprocessing	$O(l^2)$
Grouping	$O(l^3)$
Design pattern instances recovery	$O(l^3)$

Table 1. Complexity asymptotic analysis.

Based on the results from Table 1, we can conclude that the overall worst time complexity of our approach is $O(max\{n, l^3\})$. As, in a large real software system, usually $l^3 > n$, the overall complexity is $O(l^3)$.

As a conclusion, we can summarize the advantages of the search based approach proposed in this paper in comparison with existing approaches:

- The overall worst time complexity ($O(l^3)$) of our approach is reduced in comparison with the worst time complexity of a brute force approach ($O(l^{nc_p})$) (as the number of classes nc_p of classes contained in a design pattern p is greater or equal to 3).
- Our approach is not dependent on a particular design pattern. It may be used to identify instances of various design patterns, as any design pattern can be described according to the theoretical model introduced in Section 3.
- Our approach may be used to identify both *structural* and *behavioral* design patterns, as the constraints can express both structural and behavioral aspects of the application classes from the analyzed software system.

7. Conclusions and future work

We have introduced in this paper a search based approach for identifying instances of design patterns in existing software systems. We have emphasized the advantages of our approach in comparison with existing approaches in the field.

Further work can be done in the following directions:

- Improving the **Preprocessing** and **Grouping** steps from our approach.
- Applying the proposed approach on real software systems.
- Extending the proposed approach towards identifying several design patterns.
- Extending the proposed approach towards introducing design patterns in existing software systems.

References

[1] G. Antoniol, R. Fiutem, and L. Cristoforetti, *Using metrics to identify design patterns in object-oriented software*, Proc. of the Fifth International Symposium on Software Metrics - METRICS'98, 1998, pp. 23–34.

[2] James M. Bieman and Byung-Kyoo Kang, *Measuring design-level cohesion*, Software Engineering **24** (1998), no. 2, 111–124.

[3] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, *Design patterns: Elements of reusable object oriented software*, Addison-Wesley Publishing Company, USA, 1995.

[4] M. Meyer, J. Niere, and L. Wendehals, *User-driven adaption in rule-based pattern recognition*, Technical Report TR-RI-04-249 (2004), University of Paderborn, Paderborn, Germany.

[5] A. K. Jain, M. N. Murty, and P. J. Flynn, *Data clustering: a review*, ACM Computing Surveys **31** (1999), no. 3, 264–323.

[6] Kostas Kontogiannis, Renato de Mori, Ettore Merlo, M. Galler, and Morris Bernstein, *Pattern matching for clone and concept detection*, Automated Software Engineering **3** (1996), no. 1/2, 77–108.

[7] Christian Kramer and Lutz Prechelt, *Design recovery by automated search for structural design patterns in object-oriented software*, WCRE '96: Proceedings of the 3rd Working Conference on Reverse Engineering (WCRE '96), 1996, pp. 208–215.

[8] Jörg Niere, Wilhelm Schäfer, Jörg P. Wadsack, Lothar Wendehals, and Jim Welsh, *Towards pattern-based design recovery*, ICSE '02: Proceedings of the 24th International Conference on Software Engineering, 2002, pp. 338–348.

[9] Jörg Niere, Jörg P. Wadsack, and Lothar Wendehals, *Handling large search space in pattern-based reverse engineering*, IWPC '03: Proceedings of the 11th IEEE International Workshop on Program Comprehension, 2003, pp. 274.

[10] Elaine Rich and Kevin Knight, *Artificial intelligence*, 2nd ed., McGraw Hill, New York, 1991.

[11] G. Serban and I.G. Czibula, *On evaluating software systems design*, Studia Universitatis "Babes-Bolyai", Informatica **LII** (2007), no. 1, 55–66.

[12] Frank Simon, Silvio Loffler, and Claus Lewerentz, *Distance based Cohesion Measuring*, Proceedings of the 2nd European Software Measurement Conference (FESMA), 1999, pp. 69–83.

[13] L. Wendehals, *Improving Design Pattern Instance Recognition by Dynamic Analysis*, Proc. of the ICSE 2003 Workshop on Dynamic Analysis (WODA), 2003, pp. 29–32.

Department of Computer Science, Babeş-Bolyai University 1, M. Kogălniceanu Street, 400084, Cluj-Napoca, Romania,
E-mail address: gabis@cs.ubbcluj.ro

Department of Computer Science, Babeş-Bolyai University, 1, M. Kogălniceanu Street, 400084, Cluj-Napoca, Romania,
E-mail address: istvanc@cs.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LIII**, Number 1, 2008

A METHOD FOR MINING FUNCTIONAL DEPENDENCIES IN RELATIONAL DATABASE DESIGN USING FCA

KATALIN TUNDE JANOSI RANCZ AND VIORICA VARGA

Abstract. Formal Concept Analysis (FCA) is a useful tool to explore the conceptual knowledge contained in a database by analyzing the formal conceptual structure of the data. In this paper, we present a new method to optimize and extend a previous research on FCA and databases, by analyzing the functional dependencies in order to correctly build database schemata. Our method intends to mine functional dependencies in a relational database table. The novelty of our method is that it builds inverted index files in order to optimize the construction of the formal context of functional dependencies.

1. Introduction

From a philosophical point of view a concept is a unit of thoughts consisting of two parts, the extension, which are objects and the intension consisting of all attributes valid for the objects of the context. Formal Concept Analysis (FCA) introduced by [7] gives a mathematical formalization of the concept notion. A detailed mathematic foundation of FCA can be found in [3]. Formal Concept Analysis is applied in many different realms like psychology, sociology, computer science, biology, medicine and linguistics.

Formal Concept Analysis has been proved to be a valuable tool to represent the knowledge contained in a database, for instance logical implications in datasets. The subject of detecting functional dependencies in relational tables was studied in detail with different mathematical theories. Hereth (2002) presents the relationship between FCA and functional dependencies. He introduces the formal context of functional dependencies. In this context, implications hold for functional dependencies. Baixeries (2004) gives an interesting framework to mine functional dependencies using Formal Context Analysis. Detecting functional dependencies seems to be an actual theme, see [8].

Received by the editors: April 8, 2008.

2000 *Mathematics Subject Classification.* 03G10.

1998 *CR Categories and Descriptors.* H.2.4 [**Database Management**]: Systems – *Relational databases*.

This paper presents how some basic concepts from database theory translate into the language of Formal Concept Analysis and attempts to develop the functional dependencies.

We optimize an existing method introduced by [4], which provides a direct translation from relational databases into the language of power context families and calculates the functional dependencies in a relational table. In order to generate the tuple pairs necessary to build the formal context of functional dependencies we build inverted index files for the values of every attribute, because in the context of functional dependencies we need the tuple pairs, where at least one of the attribute values are common. In order to make the algorithm faster we reduce the size of the context file, which leads to fewer concepts in the lattice with useless dependencies.

2. Functional dependencies

The most used database model is the relational model. To give the structure of data [1] presents the unnamed and named perspective. The data is stored in data tables, which have attributes as columns and tuples as rows. In the named perspective the attributes are given by names, in the unnamed perspective they are given only by position. The operations on the relational model are based on algebra or on logic. The data integrity constraints of the model appear as functional dependencies.

We give the definition of a relational database from the unnamed perspective.

Definition 1. We define a relational database to be a tuple $\mathcal{D} := (\mathbf{dom}, \mathcal{N})$ with **dom** being the domain of the database and $\mathcal{N}$ being the set of named data tables in the database. A data table is any element $T \in \cup_{i \in \mathbb{N}_0} \beta \left(dom^i \right)$. The arity of T is the smallest $i \in \mathbb{N}_0$ such that $T \in \beta \left(dom^i \right)$ and is written $arity(T)$. For a tuple $t \in T$ we write $t\,[j]$ with $1 \leq j \leq arity\,(T)$ to denote the j-th value of the tuple.

Example 1. In database implementations the named perspective is used. The database scheme is composed of the table names with their attribute names. The example gives the relational scheme of a university database. Students are divided in groups; there can be many groups in one specialization. Students are marked at different disciplines.

```
Specialization [SpecID, SpecName, Language]
Groups [GroupID, SpecID]
Students [StudID, GroupID, StudName, Email]
Disciplines [DiscID, DName, CreditNr]
Marks [StudID, DiscID, Mark]
```

In this example $\mathcal{N}$ is composed of the named data tables: Specialization, Groups, Students, Disciplines, Marks, **dom** is the set of all attribute's values of the tables. In case of each table the *arity* is the number of its attributes, so Groups table has arity 2, Students has 4. Let t be a tuple (row) of table Students, than $t[1]$ is the value of tuple t for StudID, $t[2]$ is the value of GroupID in the corresponding row, so on.

In relational database design the normalization theory is used to avoid redundancy. Normal forms use the notion of functional dependencies. The following definition uses the projection relational operator, see any database theory book [1], [6]. In the following definition we use the unnamed perspective, so the attributes are given by there number.

Definition 2. Let T be a data table and $X, Y \subseteq \mathbb{N}_0$. Then, T fulfills the functional dependency $D : X \rightarrow Y$, if for all tuples $s, t \in T$ $\pi_X(s) = \pi_X(t)$ implies that also $\pi_Y(s) = \pi_Y(t)$.

Example 2. Let be the next relational table:

`StudentInfos [StudID, StudName, Email, GroupID, SpecName]`

For all tuples $s, t \in$ `StudentInfos` for which $\pi_{GroupID}(s) = \pi_{GroupID}(t)$ implies that $\pi_{SpecName}(s) = \pi_{SpecName}(t)$.

So, the following functional dependencies hold:

$$GroupID \rightarrow SpecName$$

This means, we repeat for every student from a group the specialization name.

In the same manner we can see that:

$$StudID \rightarrow StudName, Email, GroupID, SpecName$$

$$Email \rightarrow StudID, StudName, SpecName, GroupID$$

In order to define the functional dependencies in a formal context, we need the notion of Power Context Family.

Definition 3. (Power Context Family). A power context family $\overrightarrow{\mathbb{K}} := (\mathbb{K}_n)_{n \in \mathbb{K}_0}$ is a family of formal contexts $\mathbb{K}_k := (G_k, M_k, I_k)$ such that $G_k \subseteq (G_0)^k$ for $k = 1, 2, ...$ The formal contexts $\mathbb{K}_k$ with $k \geq 1$ are called relational contexts. The power context family $\overrightarrow{\mathbb{K}}$ is said to be limited of type $n \in \mathbb{N}_0$ if $\overrightarrow{\mathbb{K}} = (\mathbb{K}_0, \mathbb{K}_1, ..., \mathbb{K}_n)$, otherwise, it is called unlimited.

The following definition from [4] gives the method to construct the power context family resulting from a relational database.

Definition 4. The power context family $\overrightarrow{\mathbb{K}}(\mathcal{D})$ resulting from the canonical database translation of the relational database $\mathcal{D} = (\mathbf{dom}, N)$ is constructed in the following way: we set $\mathbb{K}_0 := (\mathbf{dom}, \emptyset, \emptyset)$ and, for $k \geq 1$, let G_k be the set of all k-ary tuples and $M_k \subseteq N$ be the set of all named data tables of arity k. The relation I_k is defined by $(g, m) \in I_k :\Leftrightarrow g \in m$.

$\mathbb{K}_2$	`Groups`
(531, I)	X
(532, I)	X
(111, M)	X
(...)	...

Table 1. $\mathbb{K}_2$ for **Example** 1

$\mathbb{K}_3$	`Specialization`	`Disciplines`	`Marks`
(M, Mathematics, German)	X		
(I, Informatics, English)	X		
(...)	...		
(11, Databases, 6)		X	
(22, Algebra, 6)		X	
(...)		...	
(101, 22, 9)			X
(157, 22, 8)			X
(...)			...

Table 2. $\mathbb{K}_3$ for **Example** 1

Example 3. Let us construct the power context family of our Example 1. $\mathbb{K}_0$ has no attributes, because we haven't any 0-ary relation. $\mathbb{K}_2$ has in his one attribute table `Groups`, this is the only table with arity 2, objects are tuples from this table, see Table 1. The attributes of $\mathbb{K}_3$ are the tables with arity 3, objects are rows from these tables, the incidence relation shows which tuple to which table belongs (Table 2). $\mathbb{K}_4$ has one attribute, `Students` table name, see Table 3.

The formal context of functional dependencies is defined in the following way in [4].

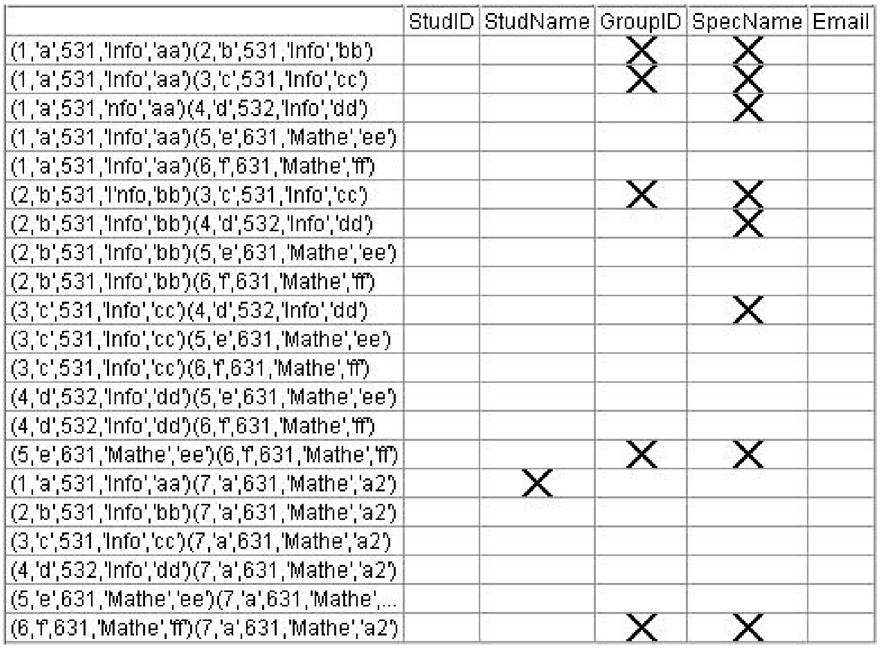
Definition 5. Let $\overrightarrow{\mathbb{K}}$ be a power context family, and let $m \in M_k$ be an attribute of the k-th context. Than the formal context of functional dependencies of m with regard to $\overrightarrow{\mathbb{K}}$ is defined as $FD\left(m, \overrightarrow{\mathbb{K}}\right) := \left(m^{I_k} \times m^{I_k}, \{\ 1, 2, ..., k\}, J\right)$ with $((g,h), i) \in J :\Leftrightarrow \pi_i(g) = \pi_i(h)$ with $g, h \in m^{I_k}$ and $i \in \{\ 1, 2, ..., k\}$.

Example 4. Let us construct the formal context of functional dependencies for table `StudentInfos` of Example 2 notated by $FD\left(StudentInfos, \overrightarrow{\mathbb{K}}(Uni)\right)$. This is a relation from database with name Uni. It has arity 5, so it is an attribute of the $\mathbb{K}_5$ from the corresponding power context family $\overrightarrow{\mathbb{K}}(Uni)$.

$\mathbb{K}_4$	`Students`
(101, 531, Irene Cates, IreneCates@email.com)	X
(157, 111, Maria Jillian,MariaJillian@yahoo.com,)	X
(234, 532, Frank Orlando,FrankOrlando@email.com)	X
(...)	...

TABLE 3. $\mathbb{K}_4$ for **Example** 1

The attributes of $FD\left(StudentInfos, \overrightarrow{\mathbb{K}}(Uni)\right)$ are the attributes of table `StudentInfos`, the objects are pairs of tuples from this table. The incidence relation of the context shows that the attribute is common to the tuple pair from the row, see Fig. 1. We give short names to students and emails, in order to fit the picture in page.



	StudID	StudName	GroupID	SpecName	Email
(1,'a',531,'Info','aa')(2,'b',531,'Info','bb')			X	X	
(1,'a',531,'Info','aa')(3,'c',531,'Info','cc')			X	X	
(1,'a',531,'nfo','aa')(4,'d',532,'Info','dd')				X	
(1,'a',531,'Info','aa')(5,'e',631,'Mathe','ee')					
(1,'a',531,'Info','aa')(6,'f',631,'Mathe','ff')					
(2,'b',531,'I'nfo,'bb')(3,'c',531,'Info','cc')			X	X	
(2,'b',531,'Info','bb')(4,'d',532,'Info','dd')				X	
(2,'b',531,'Info','bb')(5,'e',631,'Mathe','ee')					
(2,'b',531,'Info','bb')(6,'f',631,'Mathe','ff')					
(3,'c',531,'Info','cc')(4,'d',532,'Info','dd')				X	
(3,'c',531,'Info','cc')(5,'e',631,'Mathe','ee')					
(3,'c',531,'Info','cc')(6,'f',631,'Mathe','ff')					
(4,'d',532,'Info','dd')(5,'e',631,'Mathe','ee')					
(4,'d',532,'Info','dd')(6,'f',631,'Mathe','ff')					
(5,'e',631,'Mathe','ee')(6,'f',631,'Mathe','ff')			X	X	
(1,'a',531,'Info','aa')(7,'a',631,'Mathe','a2')		X			
(2,'b',531,'Info','bb')(7,'a',631,'Mathe','a2')					
(3,'c',531,'Info','cc')(7,'a',631,'Mathe','a2')					
(4,'d',532,'Info','dd')(7,'a',631,'Mathe','a2')					
(5,'e',631,'Mathe','ee')(7,'a',631,'Mathe',...					
(6,'f',631,'Mathe','ff')(7,'a',631,'Mathe','a2')			X	X	

FIGURE 1. $FD\left(StudentInfos, \overrightarrow{\mathbb{K}}(Uni)\right)$

In order to mine functional dependencies in the context defined in definition 5, we need the following proposition (see [4]).

Proposition 1. Let $\mathcal{D}$ be a relational database and m a k-ary table in $\mathcal{D}$. For two sets $X, Y \subseteq \{\ 1, ..., k\}$ we have the following equality: The columns

Y are functionally dependent from the columns X if and only if $X \to Y$ is an implication in $FD\left(m, \overrightarrow{K}(\mathcal{D})\right)$.

Example 5. Let us construct the conceptual lattice for

$$FD\left(StudentInfos, \overrightarrow{\mathbb{K}}(Uni)\right)$$

using the Concept Explorer (ConExp) from site http://sourceforge.net, see Fig. 2. We can see the implication in the context of functional dependencies, the software shows us too:

$$GroupID \to SpecName$$

This is a transitive functional dependency, so the table isn't in 3NF. The software also find the following functional dependencies:

$$StudID \to StudName, Email, GroupID, SpecName$$

$$Email \to StudID, StudName, SpecName, GroupID$$

where the right hand side (*StudID* and *Email*) are candidate keys of the table `StudInfos`.

3. Method description

This section presents how our method constructs the context of functional dependencies of a database table.

In the first step, we introduce the structure of the table to be designed and some significant tuples. It is not necessary to use all the rows of a database table, but it is important to select varied tuples, with different styles of data, in order to get as many conclusions as possible. Using definitions 3, 4, 5 we construct the formal context of functional dependencies to find existing functional dependencies as implications in the constructed table. In order to optimize the construction of the formal context, we build inverted index files for the values of every attribute. With inverted indexes the number of rows in the data file of formal context resulted by our method is half of the same value resulted using Hereth's method in [4]. In the consequence we can reduce the time to build the conceptual lattice for functional dependencies and we can eliminate useless dependencies.

An inverted index (or inverted file) is an index data structure, a sequence of (*key*, *pointer*) pairs where each pointer points to a *record* in a database which contains the key value in some particular field. The inverted index is a central component of a typical search engine indexing algorithm. For databases in which the records may be searched based on more than one field, multiple indices may be created.

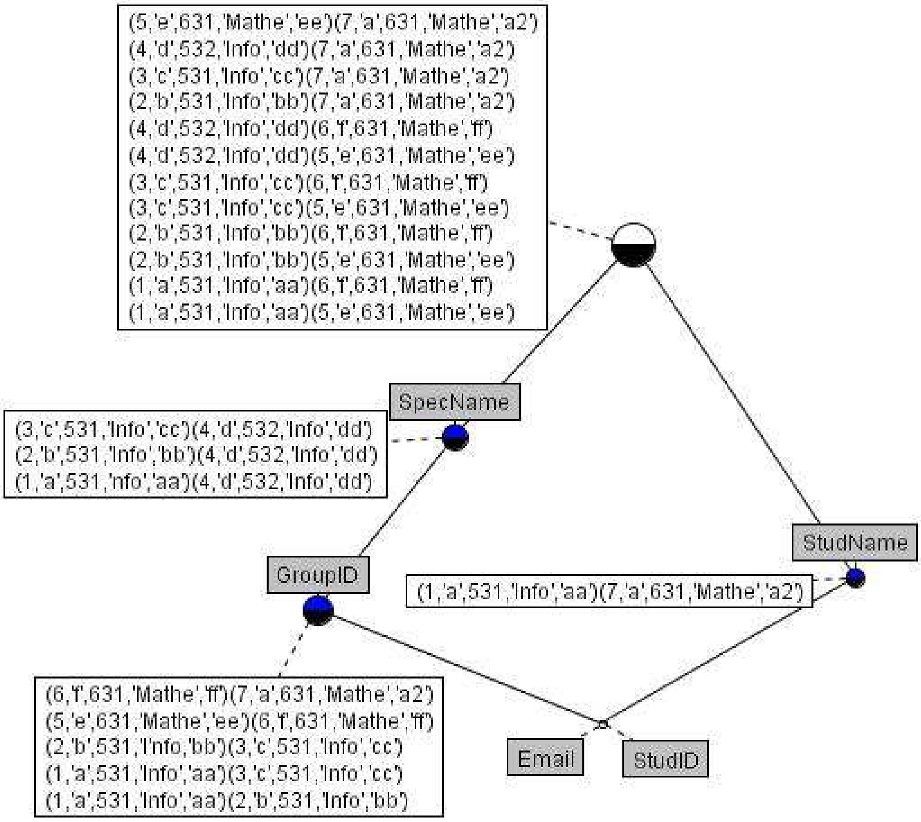

(5,'e',631,'Mathe','ee')(7,'a',631,'Mathe','a2')
(4,'d',532,'Info','dd')(7,'a',631,'Mathe','a2')
(3,'c',531,'Info','cc')(7,'a',631,'Mathe','a2')
(2,'b',531,'Info','bb')(7,'a',631,'Mathe','a2')
(4,'d',532,'Info','dd')(6,'f',631,'Mathe','ff')
(4,'d',532,'Info','dd')(5,'e',631,'Mathe','ee')
(3,'c',531,'Info','cc')(6,'f',631,'Mathe','ff')
(3,'c',531,'Info','cc')(5,'e',631,'Mathe','ee')
(2,'b',531,'Info','bb')(6,'f',631,'Mathe','ff')
(2,'b',531,'Info','bb')(5,'e',631,'Mathe','ee')
(1,'a',531,'Info','aa')(6,'f',631,'Mathe','ff')
(1,'a',531,'Info','aa')(5,'e',631,'Mathe','ee')
SpecName
(3,'c',531,'Info','cc')(4,'d',532,'Info','dd')
(2,'b',531,'Info','bb')(4,'d',532,'Info','dd')
(1,'a',531,'nfo','aa')(4,'d',532,'Info','dd')
StudName
GroupID
(1,'a',531,'Info','aa')(7,'a',631,'Mathe','a2')
(6,'f',631,'Mathe','ff')(7,'a',631,'Mathe','a2')
(5,'e',631,'Mathe','ee')(6,'f',631,'Mathe','ff')
(2,'b',531,'I'nfo,'bb')(3,'c',531,'Info','cc')
(1,'a',531,'Info','aa')(3,'c',531,'Info','cc')
(1,'a',531,'Info','aa')(2,'b',531,'Info','bb')
Email
StudID

FIGURE 2. Conceptual lattice for $FD\,(StudentInfos, \rightarrow(Uni))$

Value	Row numbers
v_{1j}	rnr^1_{1j}, rnr^2_{1j}, ...
v_{2j}	rnr^1_{2j}, rnr^2_{2j}, ...
...	
v_{mj}	rnr^1_{mj}, rnr^2_{mj}, ...

TABLE 4. Inverted index InvInd_j

We use the following notations for the j-th inverted file, which contains the different values of the attribute a_j: v_{1j}, v_{2j}, ..., v_{mj} and for each value the row numbers, where the corresponding attribute value appears, see Table 3.

Rownrs	*StudID*	*StudName*	*GroupID*	*SpecName*	*Email*
1	1	a	531	Info	aa
2	2	b	531	Info	bb
3	3	c	531	Info	cc
4	4	d	532	Info	dd
5	5	e	631	Mathe	ee
6	6	f	631	Mathe	ff
7	7	a	631	Mathe	a2

Table 5. Table StudentInfos

StudID	
value	row nrs
1	1
2	2
3	3
4	4
5	5
6	6
7	7

StudName	
value	row nrs
a	1,7
b	2
c	3
d	4
e	5
f	6

GroupID	
value	row nrs
531	1,2,3
532	4
631	5,6,7

SpecName	
value	row nrs
Info	1,2,3,4
Mathe	5,6,7

Email	
value	row nrs
aa	1
bb	2
cc	3
dd	4
ee	5
ff	6
a2	7

Table 6. Inverted index files for table StudentInfos

The context of functional dependencies being constructed, we build the concept lattice. In the top of the concept lattice will be tuple pairs, in which are no common values of the corresponding attributes, so we can omit to generate these pairs of tuples. Pairs of form (t, t), where t is a tuple of the table, have all attributes in common, these objects will arrive in the bottom of the lattice, so they can be omitted too, because they don't change the implications in the context. Finally from the resulted context we generate the functional dependencies.

Example 6. There is a simple example on how to build inverted index file. Let be the following rows in table `StudentInfos` and the inverted index files for every attribute, see Table 5 and 6.

The previous considerations alow us to formulate the next algorithm to build the context of functional dependencies, inverted index files being constructed in the same time.

```
Algorithm
for each inserted row in table $T$ do
  begin
    let $k$ be the number of row
    let $e_{k1}, e_{k2}, ..., e_{kn}$ be the attribute values of row $k$
    for j:=1 to n do // for every attribute value
      begin
        search $e_{kj}$ in the $j$-th inverted index file //search the attribute
                                    // value in the corresponding inverted file
        if find, let $v_{lj}$ be the value in the inverted file
              such that $e_{kj} = v_{lj}$ then
          begin // $k$ is added to the list of row numbers for value $v_{lj}$
            build the array list $al_{kj} = \{rnr^1_{lj}, rnr^2_{lj}, ...\}$
            add $k$ in $al_{kj}$
            add $k$ in the $j$-th inverted index in the list of row numbers
                  for value $v_{lj}$
          end
        else //value $e_{kj}$ doesn't exist in the corresponding inverted index
            //we insert it, $k$ is the first row with value $e_{kj}$ of attribute $j$
          insert new line in the $j$-th inverted index file with values $(e_{kj}, k)$
      end
    // In order to insert tuple pairs as rows in the cex file:
    build $al_k = \bigcup_{j=1}^{n} al_{kj}$          //$al_k$ contains the row numbers,
                                 // which have attributes in common with row $k$
    //if $al_k$ is empty, no row will be inserted in cex file
    if $al_k \neq \emptyset$ then
      for $s = 1$ to count($al_k$) do
        insert in cex file tuple ($k$, $al_k(s)$)
  end
```

Example 7. Let be the next table:

```
StudMarks [StudID, StudName, GroupID, Email, SpecID,
  SpecName, Language, DiscID, DName, CreditNr, Mark]
```

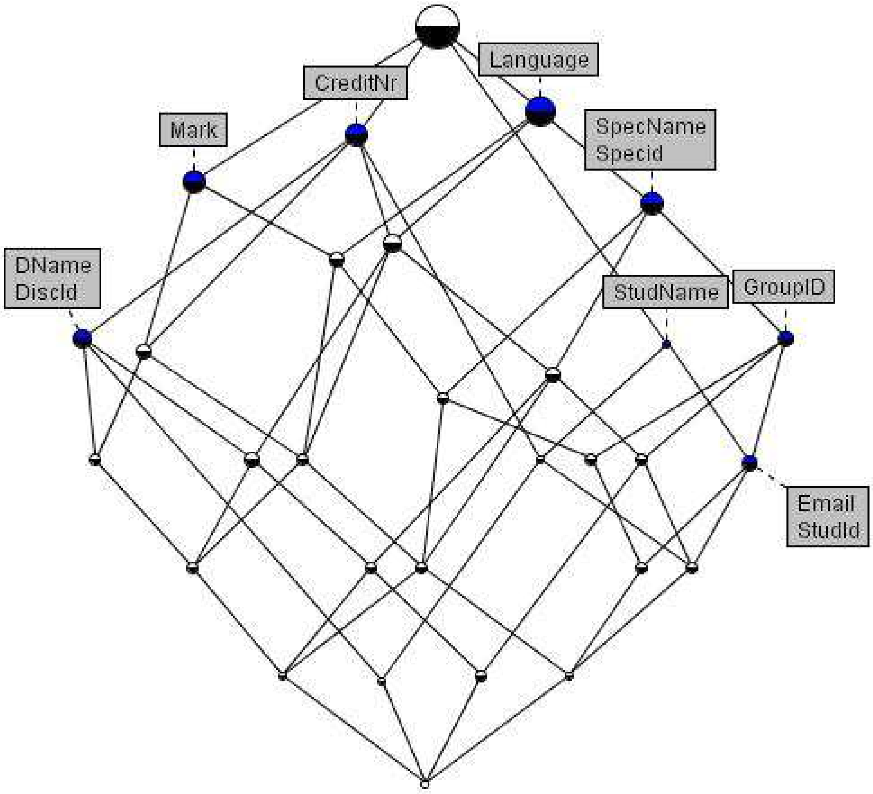


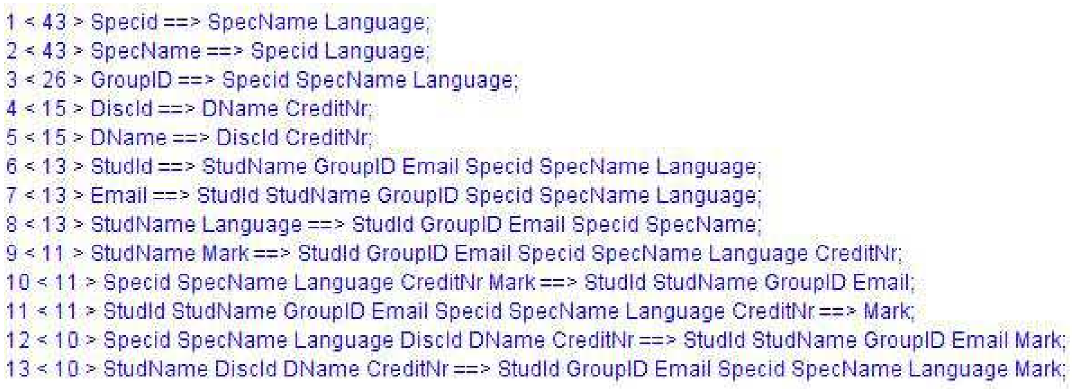
FIGURE 3. Conceptual lattice for $FD\left(StudMarks, \overrightarrow{\mathbb{K}}(Uni)\right)$ with 92 rows in the table StudMarks

In Fig. 3 is the concept lattice of the functional dependencies context for the table `StudMarks` with 92 rows. The implications in this lattice, which are functional dependencies in the table can be seen as follows: the concept with label `GroupID` is a subconcept of concept with labels `SpecID` and `SpecName.` This means, in every tuple pair where the `GroupID` field has the same value, the specialization ID and name is the same. So we have the following implications, which are functional dependencies:

GroupID →*SpecID, SpecName*

In the same manner the following functional dependencies can be read from the conceptual lattice:

DiscID →*CreditNr*
DName →*CreditNr*



```
1 < 43 > Specid ==> SpecName Language;
2 < 43 > SpecName ==> Specid Language;
3 < 26 > GroupID ==> Specid SpecName Language;
4 < 15 > DiscId ==> DName CreditNr;
5 < 15 > DName ==> DiscId CreditNr;
6 < 13 > StudId ==> StudName GroupID Email Specid SpecName Language;
7 < 13 > Email ==> StudId StudName GroupID Specid SpecName Language;
8 < 13 > StudName Language ==> StudId GroupID Email Specid SpecName;
9 < 11 > StudName Mark ==> StudId GroupID Email Specid SpecName Language CreditNr;
10 < 11 > Specid SpecName Language CreditNr Mark ==> StudId StudName GroupID Email;
11 < 11 > StudId StudName GroupID Email Specid SpecName Language CreditNr ==> Mark;
12 < 10 > Specid SpecName Language DiscId DName CreditNr ==> StudId StudName GroupID Email Mark;
13 < 10 > StudName DiscId DName CreditNr ==> StudId GroupID Email Specid SpecName Language Mark;
```

FIGURE 4. Implications, namely functional dependencies in the table StudMarks

$$\begin{array}{l} \mathit{SpecID} \rightarrow \mathit{Language} \\ \mathit{SpecName} \rightarrow \mathit{Language} \\ \mathit{StudID} \rightarrow \mathit{StudName} \\ \mathit{Email} \rightarrow \mathit{StudName} \end{array}$$

The implications in this lattice given by Conexp can be seen in Fig. 4.

4. Conclusions and further research

This paper presents a method to optimize and to reduce the workload of the users in the process of mining functional dependencies in a relational database. Our method proves that FCA visualization makes easier to manage database schemata and normal forms.

Our goal for further research is to develop an FCA exploration software based on this method to be a front-end to relational database of any content. This software would read any type of database table and construct the context of functional dependencies, this way would not be necessary to introduce the tuples by hand. As a part of this solution, we would use the resulted implications to propose the structure of the database tables.

References

[1] Abiteboul, S., Hull, R., Vianu, V.: Foundations of databases. Addison-Wesley, Reading - Menlo - New York (1995)

[2] Baixeries, J.: A formal concept analysis framework to mine functional dependencies, Proceedings of Mathematical Methods for Learning, (2004).

[3] Ganter, B., Wille, R.: Formal Concept Analysis. Mathematical Foundations. Springer, Berlin-Heidelberg-New York. (1999)

[4] Hereth, J.: Relational Scaling and Databases. Proceedings of the 10th International Conference on Conceptual Structures: Integration and Interfaces LNCS 2393, Springer Verlag (2002) 62–76

[5] Priss, U.: Establishing connections between Formal Concept Analysis and Relational Databases. Dau; Mugnier; Stumme (eds.), Common Semantics for Sharing Knowledge: Contributions to ICCS, (2005) 132–145

[6] Silberschatz, A., Korth, H. F.,Sudarshan, S.: Database System Concepts, McGraw-Hill, Fifth Edition, (2005)

[7] Wille, R. : Restructuring lattice theory: an approach based on hierarchies of concepts. In: I.Rival (ed.): Ordered sets. Reidel, Dordrecht-Boston, (1982) 445–470

[8] Yao, H., Hamilton, H. J.: Mining functional dependencies from data, Data Mining and Knowledge Discovery, Springer Netherlands, (2007)

[9] Serhiy A. Yevtushenko: System of data analysis "Concept Explorer". (In Russian). Proceedings of the 7th National Conference on Artificial Intelligence KII-2000, p. 127-134, Russia, 2000.

Sapientia University, Tg-Mures, Romania
E-mail address: `tsuto@ms.sapientia.ro`

Babes-Bolyai University, Cluj, Romania
E-mail address: `ivarga@cs.ubbcluj.ro`

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LIII**, Number 1, 2008

EVOLUTIONARY OPTIMISATION OF KERNEL FUNCTIONS FOR SVMS

LAURA DIOŞAN, ALEXANDRINA ROGOZAN, AND JEAN-PIERRE PECUCHET

Abstract. The kernel-based classifiers use one of the classical kernels, but the real-world applications have emphasized the need to consider a new kernel function in order to boost the classification accuracy by a better adaptation of the kernel function to the characteristics of the data. Our purpose is to automatically design a complex kernel by evolutionary means. In order to achieve this purpose we develop a hybrid model that combines a Genetic Programming (GP) algorithm and a kernel-based Support Vector Machine (SVM) classifier. Each GP chromosome is a tree encoding the mathematical expression of the kernel function. The evolved kernel is compared to several human-designed kernels and to a previous genetic kernel on several datasets. Numerical experiments show that the SVM embedding our evolved kernel performs statistically better than standard kernels, but also than previous genetic kernel for the considered classification problems.

1. Introduction

The general problem of Machine Learning is to search a, usually very large, space of potential hypotheses to determine the one that will best fit the data and any prior knowledge. In 1995, Support Vector Machines (SVMs) marked the beginning of a new era in the paradigm of learning from examples. Rooted to the Statistical Learning Theory and the Structural Risk Minimization principle developed by Vladimir Vapnik at AT&T in 1963 [19, 20], SVMs gained quickly attention from the Machine Learning community due to a number of theoretical and computational merits.

SVMs are a group of supervised learning methods that can be applied to classification or regression. SVMs arose from statistical learning theory; the aim being to solve only the problem of interest without solving a more difficult problem as an intermediate step. SVMs are based on the structural risk

Received by the editors: September 10, 2007.

2000 *Mathematics Subject Classification.* 68T05,91E45.

1998 *CR Categories and Descriptors.* code I.2.6 [**Learning**]: – *Concept learning.*

minimisation principle, closely related to regularisation theory. This principle incorporates capacity control to prevent over-fitting and thus is a partial solution to the bias-variance trade-off dilemma.

One issue with SVMs is finding an appropriate positive definite kernel (and its parameters) for the given data. A wide choice of kernels already exists. Many data or applications may still benefit from the design of particular kernels, adapted specifically to a given task (i.e. kernels for vectors, kernels for strings, kernels for graphs, Fisher kernels or rational kernels). There are only some hints for working with one or another of these classic kernels, because there is no rigorous methodology to choose *a priori* the appropriate one between them. Moreover, the kernel parameters influence the performance of the SVM algorithm. The selection of the penalty error for an SVM (that controls the trade-off between maximizing the margin and classifying without error) is also critical in order to obtain good performances. Therefore, one has to optimise the kernel function, the kernel parameters and the penalty error of the SVM algorithm in order to guarantee the robustness and the accuracy of an SVM algorithm. Chapelle [4] has proposed to denote the kernel and SVM parameters as hyper-parameters.

On the other hand, the evolutionary algorithms are able to search in a continuous space without respecting some conditions (requirements) as those regarding the differentiable of the score function. We do not have the certitude that the solution provided by an EA is the optimal one, but it is very close to the optimal one. The solutions proposed by an evolutionary algorithm allow for better SVM performances.

Therefore, we choose to use the evolutionary framework in order to discover the optimal expression of a new kernel and its parameters for several classification problems. The best (adapted) kernel is learnt by the algorithm itself by using the data of a particular problem. For this aim the Genetic Programming (GP) technique [12] is combined with an SVM algorithm [6, 11, 19, 15] within a two-level hybrid model. The GP-kernel is involved into a standard SVM algorithm to be trained in order to solve a particular classification problem. The optimal expression of a kernel is discovered by involving a guided search process based on genetic operations: the selection has to provide high reproductive chances to the fittest kernels, the crossover has to enable kernel-children to inherit quickly beneficial characteristics of their kernel-parents and the mutation has to ensure the diversity of the population and the exploration of the search space. After an iterative process, which runs more generations, an optimal evolutionary kernel (eK) is provided.

The paper is organized as follows: Section 2 outlines the theory behind SVM classifiers with a particular emphasis on the kernel functions. Section 3 describes the hybrid technique used in order to evolve the SVM kernels. This is followed by a special section (Section 4) where the results of the experiments are presented. Section 5 describes some related work in the field of automated generation of kernel functions. Finally, Section 6 concludes our paper.

2. Support Vector Machine

2.1. **Generalities.** Initially, *SVM* algorithm has been proposed in order to solve binary classification problems [19]. Later, these algorithms have been generalized for multi-classes problems. Consequently, we will explain the theory behind *SVM* only on binary-labelled data.

Suppose the training data has the following form: $D = (x_i, y_i)_{i=\overline{1,m}}$, where $x_i \in \Re^d$ represents an input vector and each y_i, $y_i \in \{-1, +1\}$, the output label associated to the item x_i. *SVM* algorithm maps the input vectors to a higher dimensional space where a maximal separating hyper-plane is constructed [19]. Learning the *SVM* means (implies) to minimize the norm of the weight vector (w in Eq. (1)) under the constraint that the training items of different classes belong to opposite sides of the separating hyper-plane. Since $y_i \in \{-1, +1\}$ we can formulate this constraint as:

$$y_i(w^T x + b) \geq 1,\ \forall i \in \{1, 2, \ldots, m\}^1, \tag{1}$$

where the primal decision variables w and b define the separating hyper-plane.

The items that satisfy Eq. (1) with equality are called support vectors since they define the resulting maximum-margin hyper-planes. To account for misclassification, e.g. items that do not satisfy Eq. (1), the soft margin formulation of *SVM* has introduced some slack variables $\xi_i \in \Re$ [5].

Moreover, the separation surface has to be nonlinear in many classification problems. *SVM* was extended to handle nonlinear separation surfaces by using a feature function $\phi(x)$. The *SVM* extension to nonlinear datasets is based on mapping the input variables into a feature space $\mathcal{F}$ of a higher dimension and then performing a linear classification in that higher dimensional space. The important property of this new space is that the data set mapped by ϕ might become linearly separable if an appropriate feature function is used, even when that data set is not linearly separable in the original space.

Hence, to construct a maximal margin classifier one has to solve the convex quadratic programming problem encoded by Eq. (2), which is the primal formulation of it:

$$\begin{array}{ll} \text{minimise}_{w,b,\xi} \frac{1}{2} w^T w + C \sum_{i=1}^m \xi_i & \\ \text{subject to:} & y_i(w^T \phi(x_i) + b) \geq 1 - \xi_i, \\ & \xi_i \geq 0, \forall i \in \{1, 2, \ldots, m\}. \end{array} \tag{2}$$

[1] where v^T represent the transpose of v

The coefficient C (usually called *penalty error* or *regularization parameter*) is a tuning parameter that controls the trade off between maximizing the margin and classifying without error. Larger values of C might lead to linear functions with smaller margin, allowing to classify more examples correctly with strong confidence. A proper choice of this parameter is crucial for SVM to achieve good classification performance.

Instead of solving Eq. (2) directly, it is a common practice to solve its dual problem, which is described by Eq. (3):

$$\begin{array}{ll} \text{maximise}_{a \in \Re^m} \sum_{i=1}^{m} a_i - \frac{1}{2} \sum_{i,j=1}^{m} a_i a_j y_i y_j \phi(x_i)^T \phi(x_j) & \\ \text{subject to} \quad \sum_{i=1}^{m} a_i y_i = 0, & \\ \qquad\qquad\quad 0 \leq a_i \leq C, \forall i \in \{1, 2, \ldots, m\}. & \end{array} \tag{3}$$

In Eq. (3), a_i denotes the Lagrange variable for the i^{th} constraint of Eq. (2).

The optimal separating hyper-plane $f(x) = w \cdot \phi(x) + b$, where w and b are determined by Eq. (2) or Eq. (3) is used to classify the un-labelled input data x_k:

$$y_k = sign\left(\sum_{x_i \in S} a_i \phi(x_i)^T \phi(x_k) + b\right), \tag{4}$$

where S represents the set of support vector items x_i.

We will see in the next section that is more convenient to use a kernel function $K(x, z)$ instead of the dot product $\phi(x)^T \phi(z)$.

2.2. **Kernel formalism.** The original optimal hyper-plane algorithm proposed by Vapnik in 1963 was a linear classifier [19]. However, in 1992, Boser, Guyon and Vapnik [2] have suggested a way to create non-linear classifiers by applying the *kernel trick*. Kernel methods work by mapping the data items into a high-dimensional vector space $\mathcal{F}$, called feature space, where the separating hyper-plane has to be found [2]. This mapping is implicitly defined by specifying an inner product for the feature space via a positive semi-definite kernel function: $K(x, z) = \phi(x)^T \phi(z)$, where $\phi(x)$ and $\phi(z)$ are the transformed data items x and z [16]. Note that all we required is the result of such an inner product. Therefore we do even not need to have an explicit representation of the mapping ϕ, neither to know the nature of the feature space. The only requirement is to be able to evaluate the kernel function on all the pairs of data items, which is much easier than computing the coordinates of those items in the feature space.

The kernels that correspond to a space embedded with a dot product belong to the class of positive definite kernels. This has far-reaching consequences. The positive definite and symmetric kernels verify the Mercer's theorem [13] - a condition that guarantees the convergence of training for discriminant classification algorithms such as *SVM*s. The kernels of this kind can be evaluated efficiently even though they correspond to dot products in infinite dimensional dot product spaces. In such cases, the substitution of the dot product with the kernel function is called the *kernel trick* [2].

In order to obtain an *SVM* classifier with kernels one has to solve the following optimization problem:

$$\begin{array}{ll} \text{maximise}_{a\in\Re^m} \sum_{i=1}^{m} a_i - \frac{1}{2}\sum_{i,j=1}^{m} a_i a_j y_i y_j K(x_i, x_j) & \\ \text{subject to} \quad \sum_{i=1}^{m} a_i y_i = 0, & \\ \qquad\qquad\quad 0 \le a_i \le C, \forall i \in \{1, 2, \ldots, m\}. & \end{array} \tag{5}$$

In this case, Eq. (4) becomes:

$$y_k = sign\left(\sum_{x_i\in S} a_i K(x_i, x_k) + b\right), \tag{6}$$

where S represents the set of support vector items x_i.

There are a wide choice for a positive definite and symmetric kernel K from Eq. (6). The selection of a kernel has to be guided by the problem that must be solved. Choosing a suitable kernel function for *SVM*s is a very important step for the learning process. There are few if any systematic techniques to assist in this choice. Until now, different kernels for vectors have been proposed [18]; the most utilised of them by an *SVM* algorithm are listed in Table 1.

TABLE 1. The expression of several classic kernels.

Name	**Expression**	**Type**
Sigmoid	$K_{Sig}\ (x,z) = \tanh(\sigma x^T \cdot z + r)$	projective
RBF	$K_{RBF}(x,z) = exp(-\sigma\vert x-z\vert^2)$	radial
Polynomial	$K_{Pol}\ (x,z) = (x^T \cdot z + coef)^d$	projective

3. Evolved Kernels

This section describes our approach for automatic design of kernels. The model's idea was initially proposed in [8] and here we try to detail it and to performed a more deeply analysis of the new evolved kernels. The model is a hybrid one: it uses *GP* [12] to construct positive and symmetric kernel functions, and optimizes a fitness function by using an *SVM* classifier (see Figure 1). A *GP* chromosome provides the analytic expression of such evolved kernels. The model we describe actually seeks to replace the expert domain knowledge concerning the design of the *SVM*'s kernel function with a *GP* algorithm.

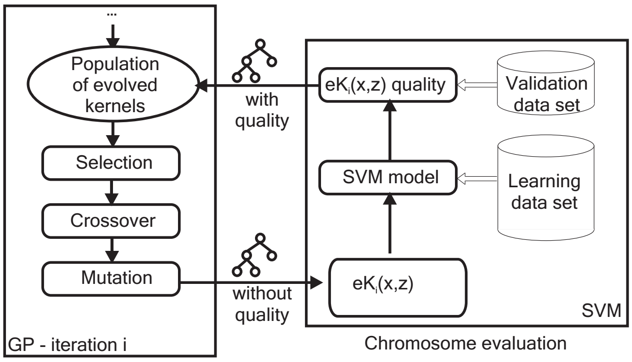


FIGURE 1. Sketch of the hybrid approach

Our hybrid model is structured on two levels: a macro level and a micro level. The macro level algorithm is a standard *GP* [12], which is used to evolve the mathematical expression of a kernel. The steady-state evolutionary model [17] is involved as an underlying mechanism for the *GP* implementation. A steady state algorithm is much more tolerant of poor offspring than a generational one. This is because in most implementations, the best individuals from a given generation will always be preserved in the next generation, giving themselves another opportunity to be selected for reproduction. The best individuals are therefore given more chances to pass on their successful traits. The *GP* algorithm starts by an initialisation step of creating a random population of individuals (seen as kernel functions). The following steps are repeated until a given number of iterations are reached: two parents are selected using a binary selection procedure; the parents are recombined in order to obtain an offspring O; the offspring is than considered for the mutation; the new individual O^* (obtained after mutation) replaces the worst individual W in the current population if O^* is better than W.

The micro level algorithm is an *SVM* classifier. It is taken from *LIBSVM* [3] library. The original implementation of the *SVM* algorithm proposed in [3] allows using several well-known kernels (Polynomial, RBF and Sigmoid – see Table 1). In the numerical experiments, a modified version of this algorithm, which is based on the evolved kernel (eK) is also used. The quality of each *GP* individual is determined by running the *SVM* algorithm, which uses the eK encoded in the current chromosome ($K_{iter,ind}$ that corresponds to the ind^{th} individual from the population during the $iter^{th}$ iteration). The accuracy rate estimated by the classifier (on the validation set) represents the fitness of the *GP* chromosome.

3.1. **Kernel representation.** In our model, the GP chromosome is a tree encoding the mathematical expression of the kernel function. Unlike a classic GP tree, our model uses a particular category of trees that can contain two types of nodes: scalar nodes and vectorial nodes. The terminal set is composed only by vectors from $\Re^d$: $VTS = \{x | x \in \Re^d\}$ (which correspond to the input data). Since a kernel function operates only on two samples the resulting terminal set VTS contains only two vector elements: x and z.

The function set (FS) contains two types of operations: scalar operations and vectorial ones. The scalar function set (*SFS*) contains several well-known binary (+, −, ×, /) and unary (sin, cos, exp, log) operators. The vectorial function set *VFS* (see Table 2) contains two types of primitive functions: operators that transform the initial multi-dimensional space into an $\mathcal{R}$ space (known as *norm* functions) and operators that preserve the dimensionality of the initial space. We also include several element-wise operations (EWOs) in this last function category.

Table 2. The vectorial function set - VFS: the norms and the element-wise operations

Type	**Elements**	**Definition**
Norms	EN	$EN(x,z) = \sum_{i=1}^{d}(x_i - z_i)^2$
	SP	$SP(x,z) = \sum_{i=1}^{d} x_i z_i$
	GN	$GN(x,z) = e^{-\gamma \times \sum_{i=1}^{d}(x_i - z_i)^2}$
EWOs	$\oplus$	$x \oplus z = v,\ v_i = x_i + z_i,\ i = \overline{1,d}$
	$\ominus$	$x \ominus z = v,\ v_i = x_i - z_i,\ i = \overline{1,d}$
	$\otimes$	$x \otimes z = v,\ v_i = x_i * z_i,\ i = \overline{1,d}$

Starting with a bottom-up tree reading, the functions operate in this way:

- the element-wise operations transform the d-dimensional space of input instances into another d-dimensional space.
- the norms (e.g., Euclidean, Gaussian, Scalar Product) transform the d-dimensional space into an one-dimensional space. The norms link the vector space with the scalar space in our GP tree.
- the scalar operations are used to combine the outputs of different norms.

Note that these EWOs are performed in a manner that preserves a valid dimension for the resulted vector. Our vectorial multiplication operation $\otimes$

is an element-wise operation; it is different from the cross product, which is a geometrical vector multiplication. The cross product performs the transformation $(\Re^d, \Re^d) \longmapsto \Re^d \times \Re^d$, or in our model we must have a transformation $(\Re^d, \Re^d) \longmapsto \Re^d$.

An example of a GP chromosome is depicted in Figure 2. The nodes that contain scalar symbols form a connex region. The rest of the nodes form one or more connex sub-regions.

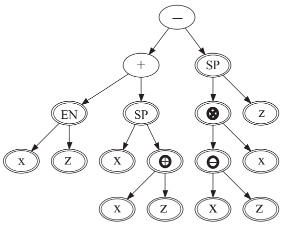


FIGURE 2. Example of a chromosome - the nodes designed by only a circle contain scalar symbols and the nodes designed by two circles contain vectorial symbols (functions and terminals). The vectors x and z, representing two data items, are the kernel inputs.

The *grow method* [1], which is a recursive procedure, is used to initialize a *GP* individual. This initialisation method is well known in the literature for its robustness. The root of each *GP* tree must be a function from FS. If a node contains a function, then its children are initialized either with another function or with a terminal. The initialization process is stopped when is attained a leaf node or at the maximal depth of the tree (the nodes from the last level will be initialised by terminals). Moreover, the maximal depth of a *GP* chromosome has to be large enough in order to assure a sufficient search space for the optimal expression of our evolutionary kernel.

3.2. **Genetic Operations.**

3.2.1. *Selection.* The selection operator chooses from the current population which individuals will act like parents in order to create the next generation of kernels. Selection has to provide high reproductive chances to the fittest kernels but, at the same time, it has to preserve the exploration of the search space. The choice of which kernels are allowed for reproducing determines which regions of the search space will be visited next. Indeed, achieving equilibrium between the exploration and the exploitation of the search space is very important for an evolutionary algorithm. When performing selection for recombination, the kernels are compared by means of a fitness function that evaluates how good a potential solution is for the given problem.

3.2.2. *Crossover.* The crossover operator assures the diversity of the kernels and is performed in a tree-structure preserving way in order to ensure the validity of the offspring: first as a mathematical expression and second as a Mercer's kernel. The idea behind crossover is that the new chromosome may be better than both of the parents if it takes the best characteristics from each of the parents.

The model we describe uses an one-cutting point crossover [12]. Having two parent trees, we randomly choose a cutting point in the first parent, another cutting-point in the second parent and then, we exchange the subtrees rooted to these points. This crossover type has been used because it is able to guarantee a quite quickly convergence of the *GP* algorithm.

The cutting-point in the first parent is chosen randomly, but the other cutting-point is constrained by the position of the first one. Thus, if the first cutting-point is chosen right above a node that contains a scalar operator, then the other cutting point must be chosen above another scalar node (from the second parent). A similar procedure must be applied if the first cutting point is chosen right above a node that contains a vector operator.

Why we need this restriction? Because if we choose the first cutting point between a scalar function and a norm, then we must replace the sub-tree whose root is the norm with a sub-tree from the other parent. If the cutting-point in the second parent is chosen right above a node that contains a scalar function, then there are two possibilities: the $\Re$ function from the first parent acts on a sub-tree rooted by a scalar function or on a sub-tree rooted by a norm. These two possibilities are correct and they ensure a valid offspring. If the second cutting-point is chosen right above a node that contain a vector operator, then it is possible that scalar function from the first parent act on a vector (which is an impossible operation).

3.2.3. *Mutation.* The purpose of the mutation operator is to create new individuals by small and stochastic perturbations of a chromosome. Mutation operator aims to achieve some stochastic variability of an evolutionary algorithm in order to get a quicker convergence. Furthermore, the mutation operator aims to produce diversity of the population of candidate solutions and to reconsider the lost genetic material of the population. Mutation is therefore responsible for exploring new promising regions of the search space and not for exploiting those already discovered. This genetic operation, also as the crossover, preserves the syntactical validity of the new individual. For a *GP*-based kernel, a cutting point is randomly chosen: the sub-tree belonging to that point is deleted and a new sub-tree is grown there by applying the same random growth process that was used to generate the initial population. Note that the maximal depth allowed for the *GP* trees limits the growth process.

3.3. **Fitness Assignment.** We must provide several information about the datasets, before to describe the chromosome evaluation. The data sample was randomly divided in two disjoint sets: a training set (80%) - for models building - and a testing set (20%) - for performances evaluation. The training set was than randomly partitioned into learning (2/3) and validation (1/3) parts.

The SVM algorithm kernel uses the learning subset to construct the SVM model and the validation subset for the evolved kernel performance assignment. The quality of an evolved kernel, which is the current GP chromosome, can be measured by the classification accuracy rate computed on the validation data set. The accuracy rate represents the number of correctly classified items over the total number of items. Note that we deal with a maximization problem: greater accuracy rates are, better evolved kernels are.

In order to evaluate the quality of a GP tree, it is also necessary to take into account if the expression encoded into the current chromosome is a valid kernel or not. We must verify therefore if the current expression satisfies the Mercer conditions [5] regarding the positivity and the symmetry of the Gram matrix associated to a kernel function. We have used the penalty method to involve these restrictions in the evaluation process. More exactly, two important steps are performed:

- (1) kernel positivity and symmetry verification - if a GP tree does not satisfy these conditions then the fitness of the GP tree will be 0; otherwise, we can go to step 2.
- (2) SVM algorithm running - there are two stages in this run: in the first stage the SVM algorithm embedding the evolved kernel is constructed by using the learning data; in the second stage, the SVM algorithm with the evolved kernel classifies the items from the validation set. The accuracy rate estimated on this subset will represent the quality of the GP chromosome.

4. Numerical experiments

In this section, several numerical experiments about evolving kernel functions for different classification problems are detailed. After evolving it on the validation set, the kernel is embedded into an SVM classifier, which is run on the test dataset. The SVM algorithm based on the classical kernels are also used to classify the test data set. Finally, the performances of different classifiers are compared.

Four data sets [9] are used in thee experiments. All the data sets contain information about the real-world problems (DS_1 and DS_2 – classification task is to determine whether a person makes over $50K/year or not, DS_3 and DS_4 – classifying whether a web page belongs to a category or not). A binary classification problem must be solved in each of these cases. The structure of the problems is presented in Table 3.

Table 3. The structures of the data sets - each dataset is split into training set and testing set. For each of these two subsets it is specified: the total number of items, the number of items from the first class and the number of items from the second class, respectively

Data	**Training**			**Testing**		
set	**Total**	**1^{th} class**	**2^{nd} class**	**Total**	**1^{th} class**	**2^{nd} class**
a1a	1604	395	1209	30995	7446	23549
a2a	2264	572	1692	30295	7269	23026
w1a	2477	2404	73	47272	45864	1408
w2a	3470	3362	108	46279	44906	1373

The steady-state model [17] is used for the *GP* algorithm. A population of 50 individuals is evolved during 50 iterations, which is a reasonable limit to assure the diversity of our *eK*s. The maximal depth of a *GP* tree is limited to 10 levels, which allows encoding till 2^{10} combinations. This maximal depth was fixed by tacking into account the bloat problem (the uncontrolled growth of programs during *GP* runs without (significant) return in terms of fitness [14]). Furthermore, several empirical tests indicated that the efficient kernel-trees do not expand to more than 10 levels. A binary tournament selection, a probabilistic crossover, and a probabilistic mutation are performed in order to obtain a new generation of chromosomes. The values of the crossover and mutation probabilities ($p_c = 0.8$ and $p_m = 0.3$) are chosen in order to assure a sufficient diversity of the population. The values used for the population size and for the number of generations have been empirically chosen based on the best results computed on the validation set. The soft margin hyper-parameter C, which weights the misclassification errors, has been set to 1 for all the classifiers used in our experiments.

4.1. **Experiment 1.** New kernel functions are evolved in this experiment. As we already mentioned, there are two different stages in this experiment: in the first stage the kernel expression is learnt and in the second stage the best evolved kernel function is tested.

We obtain various expressions of the kernel function, during different runs, all of them having a similar complexity.

Figure 3 depicts the evolution of the quality for the best evolved kernels along the number of iterations (for all the problems on the validation data sets). Only the values corresponding to the first 20 generations are depicted in this graphic for a better visualisation. Small improvements can be observed in the chromosomes quality (or in the accuracy rate) after the first 15 GP generations. This aspect is very important and it proves that the proposed model can discover an efficient kernel in only a few generations.

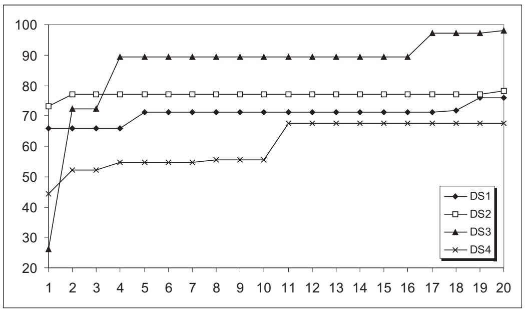


Figure 3. Evolution (on the validation set) of the best evolved kernel for all the problems.

Table 4 presents the accuracy rates estimated on the test data for each of the problem and the expression of the best evolved kernel (provided by the training process).

Table 4. The accuracy rates obtained on test datasets with an SVM algorithm that involves the best kernel expression learnt during the training stage.

Problem	**Acc**	**Kernel Expression**
DS_1	72.43	$K_1^*(x,z) = GN(z, x \ominus z) \times GN(x, z \ominus x)$
		$K_1^*(x,z) = e^{-\gamma \sum_{i=1}^{d}[z_i-(x_i-z_i)]^2} e^{-\gamma \sum_{i=1}^{d}[x_i-(z_i-x_i)]^2}$
DS_2	79.60	$K_2^*(x,z) = SP(z, x \otimes z \oplus x)$
		$K_2^*(x,z) = \sum_{i=1}^{d} z_i[x_i(z_i+x_i)]$
DS_3	89.39	$K_3^*(x,z) = GN(x,z) + SP(z,x)$
		$K_3^*(x,z) = e^{-\gamma \sum_{i=1}^{d}(x_i-z_i)^2} + \sum_{i=1}^{d} z_i x_i$
DS_4	90.27	$K_4^*(x,z) = SP(z, x \otimes z \oplus x)$
		$K_4^*(x,z) = \sum_{i=1}^{d} z_i[x_i(z_i+x_i)]$

4.2. **Experiment 2.** This experiment serves our purpose to compare the best evolved kernels to three commonly used kernels (the sigmoid kernel, the polynomial kernel and the radial basis function (RBF) kernel, respectively) and to the genetic kernel (GK) proposed in [10].

The SVM algorithm is run by the same error penalty as that from the evolving stage, but using the sigmoid kernel, the polynomial kernel, and the RBF kernel, respectively. For the Sigmoid kernel the parameter values are $\sigma = 0.01$ and $r = 0$, for the Polynomial kernel the degree d is set to 2 and for the RBF kernel the bandwidth value σ is 0.01. These values have been optimised by a parallel grid search method.

The results obtained by running the SVM with our evolved kernel (the second column), the Polynomial kernel, the RBF kernel, the Sigmoid kernel (the next three columns) and the evolved kernel described in [10] (the last column), respectively, are presented in Table 5.

The accuracy rate reflects the classification performance of the SVM algorithm in a confidence interval. The confidence intervals associated to the performances of the systems must be computed in order to decide if a system outperforms another system. If these intervals are disjoint, then one system outperforms the other ones. A confidence interval of 95% is used in order to perform a statistical examination of the results. Therefore, the probability that the accuracy estimations are not in the confidence interval is 5% (see Equation (7)).

$$\Delta I = 1.96 \times \sqrt{\frac{Acc(100 - Acc)}{N}}\%, \tag{7}$$

where N represents the number of test examples. In addition, Table 5 displays the corresponding confidence intervals (on the test set of each problem).

Table 5. The accuracy rates and their confidence intervals for each test data set using different kernel expressions.

Dataset	eK	K_{Pol}	K_{RBF}	K_{Sig}	GK
DS_1	72.43±0.50	73.40±0.49	71.28±0.50	71.28±0.50	75.62±0.48
DS_2	79.60±0.45	78.19±0.47	77.66±0.47	78.72±0.46	76.00±0.48
DS_3	89.39±0.28	74.24±0.39	84.47±0.33	37.12±0.44	88.25±0.29
DS_4	90.27±0.27	88.11±0.29	86.49±0.31	83.78±0.34	88.11±0.29

Table 5 shows that the accuracy rates computed by our hybrid approach are globally better than those computed by an SVM classifier that involves the classic kernels and also than the GK method proposed in [10] (in 3 cases out of 4). These results prove that evolving a new kernel adapted to the classification problem is more promising than computing the results by using several well-known fixed kernels and picking the best. However, more extended experiments are need in order to validate our hybrid approach. Furthermore, it is clear (Table 5) that the evolved kernels perform better (from a statistically point of view) than the simple ones (the corresponding confidence intervals are disjointed) for three problems (out of four).

5. Related Work and Discussions

Evolutionary techniques have been used in the past in order to evolve complex functions in different domains. For instance, the expression of the crossover operator used by the genetic algorithms for function optimization [7] is only an example.

Although several methods [4, 21] have been proposed for optimizing the parameters of the kernel functions, to our knowledge, only one attempt, performed by Howley and Madden [10], yields of evolving the kernel function. The authors [10] have tried to find an optimal kernel function by using the genetic programming technique also. There are several and important differences between their approach and the model we describe. These differences regard: the function set, the terminal set and the Mercer conditions.

Howley and Madden have used only a few binary operators (+, −, ×), whereas we extend the function set by adding: several unary scalar operations (exp, sin, cos, log); several norm functions (EN, SP, GN - that transform the $\Re^d$ space into an $\Re$ space and create the link between the GP tree part reserved for the $\Re^d$ space and the GP tree part reserved for the $\Re$ space) and several element-wise operations ($\oplus, \ominus, \otimes$). Moreover, they have used the same function set for both type of operations (scalar and vectorial). The model we develop uses two different function sets: one for the scalar operations and one for the vectorial ones.

The terminals can be either vectors, or scalar values in the model proposed in [10]. Our approach uses only vectors as terminals in the GP tree. By contrast a trick has been used in [10]: the model decides the type of operators (scalar or vectorial) based on the type of arguments (if both arguments are scalar then the function is a scalar one and if at least one operand is a vector, then the vectorial meaning for the operator is used) - a bottom-up approach. The current model is a top-down one: it decides the type of operands based on the type of functions.

In the model we describe, the set of functions contains both scalar and vector operators which are used in order to generate valid kernel expressions (starting by the initialization stage and continuing by the crossover and mutation steps); this is different to the Howley's approach [10] were the correctness of the kernel is imposed after the construction stage. The positivity and the symmetry of the evolved kernels learnt by our approach are verified when a kernel is evaluated (the expressions that do not satisfy these constraints are penalized). Unlike to this, in [10] the authors have first evaluated the kernel encoded into a GP tree on two samples x and z. These samples have been swapped and the kernel has been evaluated again. The dot product of these two evaluations has been returned as the kernel output. In this manner, the obtained kernels are symmetric, but there is no guarantee that they obey Mercer's theorem. Moreover, the dot-product operation has increased the kernel complexity.

Comparing to the standard SVM with a fixed kernel, the hybrid model we describe involves certainly more computations because of the additional GP step, which is needed in order to evolve an optimal kernel function. However, more computations are performed only for the training stage, whereas our hybrid model may give better accuracy when classifying the unlabelled data.

6. Conclusion and Further Work

A hybrid technique for evolving kernel functions has been described in this paper. The model has been used in order to discover and adapt (optimise) the mathematical expressions of the kernel function involved in an SVM algorithm used for binary classification problems.

We have performed several numerical experiments in order to compare our evolved kernel to others kernels (human designed or not). The numerical experiments have shown that the evolved kernels perform slightly better than the standard kernels (the sigmoid, the polynomial and the RBF kernels) or the genetic kernels proposed by Howley and Madden [10]. However, for a more pertinent conclusion regarding that the proposed method is a good one, it should be supported by a stronger statistical analysis and by a new set of experiments (especially for large data sets).

Further work will be focused on evolving better kernel functions, by taking into account different initialization strategies in order to improve the quality of the selected kernels and/or by using a multiple kernel approach.

References

[1] Banzhaf, W. Introduction. *Genetic Programming and Evolvable Machines 6*, 1 (2006), 5–6.

[2] Boser, B. E., Guyon, I., and Vapnik, V. A training algorithm for optimal margin classifiers. In *COLT* (1992), pp. 144–152.

[3] Chang, C.-C., and Lin, C.-J. *LIBSVM: a library for SVM*, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[4] Chapelle, O., Vapnik, V., Bousquet, O., and Mukherjee, S. Choosing multiple parameters for SVM. *Machine Learning 46*, 1/3 (2002), 131–159.

[5] Cortes, C., and Vapnik, V. Support-vector networks. *Machine Learning 20* (1995), 273–297.

[6] Cristianini, N., and Shawe-Taylor, J. *An Introduction to Support Vector Machines.* Cambridge University Press, 2000.

[7] Dioşan, L., and Oltean, M. Evolving the structure of the particle swarm optimization algorithms. In *EvoCOP 2006* (2006), J. Gottlieb and et al., Eds., vol. 3906 of *LNCS*, Springer, pp. 25–36.

[8] Dioşan, L., Rogozan, A., and Pécuchet, J.-P. Evolving kernel functions for SVMs by Genetic Programming. In *ICMLA'07, Ohio, USA* (2007).

[9] D.J. Newman, S. Hettich, C. B., and Merz, C. UCI repository of machine learning databases, 1998.

[10] Howley, T., and Madden, M. G. The genetic kernel SVM: Description and evaluation. *Artif. Intell. Rev 24*, 3-4 (2005), 379–395.

[11] Joachims, T. Making large–scale SVM learning practical. In *Advances in Kernel Methods — Support Vector Learning* (1999), B. Scholkopf, C. J. C. Burges, and A. J. Smola, Eds., MIT Press, pp. 169–184.

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

[13] Mercer, J. Functions of positive and negative type and their connection with the theory of integral equations. *Philosophical Transactions of the Royal Society 209* (1909), 415–446.

[14] Poli, R., Langdon, W. B., and McPhee, N. F. *A field guide to genetic programming.* Published via http://lulu.com and freely available at http://www.gp-field-guide.org.uk, 2008.

[15] Schoelkopf, B., and Smola, A. J. *Learning with Kernels.* The MIT Press, 2002.

[16] Schölkopf, B. The kernel trick for distances. In *NIPS* (2000), T. K. Leen, T. G. Dietterich, and V. Tresp, Eds., MIT Press, pp. 301–307.

[17] Syswerda, G. A study of reproduction in generational and steady state genetic algorithms. In *Proceedings of FOGA Conference* (1991), G. J. E. Rawlins, Ed., Morgan Kaufmann, pp. 94–101.

[18] Taylor, J. S., and Cristianini, N. *Kernel Methods for Pattern Analysis.* Cambridge University Press, 2004.

[19] Vapnik, V. *The Nature of Statistical Learning Theory.* Springer, 1995.

[20] Vapnik, V. *Statistical Learning Theory.* Wiley, 1998.

[21] Weston, J., Mukherjee, S., Chapelle, O., Pontil, M., Poggio, T., and Vapnik, V. Feature selection for SVMs. In *NIPS* (2000), T. K. Leen, T. G. Dietterich, and V. Tresp, Eds., MIT Press, pp. 668–674.

LITIS, EA - 4108, INSA, Rouen, France and Computer Science Department, Babeş Bolyai University, Cluj Napoca, Romania
E-mail address: lauras@cs.ubbcluj.ro

LITIS, EA - 4108, INSA, Rouen, France
E-mail address: arogozan@insa-rouen.fr

LITIS, EA - 4108, INSA, Rouen, France
E-mail address: pecuchet@insa-rouen.fr

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LIII**, Number 1, 2008

EVOLVING NETWORK TOPOLOGIES FOR CELLULAR AUTOMATA

ANCA GOG

Abstract. The problem of evolving network topologies for celular automata has been approached by means of circular evolutionary algorithms. This application is based on Watts proposal to consider small-world topologies for CAs. He has shown that small-world networks could give a better performance for problems like the density task, compared to the performance obtained when considering regular lattices for CAs. The circular evolutionary algorithm proposed in this paper has been successfully applied for evolving network topologies for the density task.

1. Introduction

A new class of evolutionary techniques called Circular Evolutionary Algorithms (CEA) is proposed. The main feature of these evolutionary algorithms is a new selection scheme according to which each individual is recombined. The philosophy behind this new model is a gradual propagation of the fittest genetic material into the population. This goal is achieved by considering and interpreting both a time dimension and a space dimension for the algorithm.

CEA selection and recombination take place asynchronously, which allows an improvement of the individuals during the process of selection and recombination in one generation. The circular settlement of all the individuals from the population according to their fitness allows us to define a new notion of neighborhood, recombination taking place only between individuals belonging to the same neighborhood. The problem of evolving networks topologies for cellular automata is addressed by using the proposed model.

Numerical experiments reported in this paper are just preliminary results referring to the performance of obtained networks. Their study and classification is the subject of future work.

Received by the editors: May 2, 2008.

2000 *Mathematics Subject Classification.* 68T20, 68Q80.

1998 *CR Categories and Descriptors.* I.2.8 [**Artificial Intelligence**]: Problem solving, Control methods, and Search – *Heuristic methods*; F.1.1 [**Computation by Abstract Devices**]: Models of Computation – *Automata*.

The paper is organized as follows. The new circular search model is described in the second section. The problem of evolving network topologies for cellular automata and existing methods are described in the third section. Results obtained after applying a circular evolutionary algorithm for this problem are presented in the fourth section. Conclusions are presented in the last section of the paper.

2. Circular Search Model

A new evolutionary model is proposed in what follows. A new way of understanding the role of the selection process is the foundation of this model. A new population topology and an asynchronous application of the search operators are the main features that arise from this new philosophy of selecting individuals for recombination. The aim of the proposed technique is to ensure a good exploitation of the good genetic material already obtained by the search process, but in the same time to allow the increase of diversity in the population. This aim is achieved by transferring to all individuals from the population genetic material that is believed to be relevant for the search process in a step by step manner that will be exhaustively explained in what follows.

Let us suppose that $P(t)$ is the current population at the time step t. The size of the population is fixed during all stages of the algorithm and is chosen to be a square number, in order to allow a certain topology of the population. Let n^2 be the size of the population (n is an even number). The algorithm ends after a certain number of generations, given as parameter of the algorithm.

2.1. **Space Dimension.** All the individuals from the population are sorted according to their fitness relative to the problem to solve. They will be distributed over $\frac{n}{2}$ concentric circles following the next constraint: the fittest individuals will be placed on the smallest circle, while the less fit individuals will be placed on the larger circle. Moreover, the number of individuals placed on circle $i (i = 0, \frac{n}{2} - 2)$ is $4(n - 2i - 1)$. This means that the individuals belonging to the concentric circles can be easily transposed into a two-dimensional grid. Figure 1 describes proposed topology using both concentric circles and the corresponding two-dimensional grid.

Let us suppose that we obtain the sorted population $P(t) = x_1, x_2, \ldots, x_{n^2}$, where x_1 is the fittest individual and x_{n^2} is the worst individual in the population. On the smallest circle are placed the fittest four individuals (x_1, x_2, x_3, x_4) from the population (their order does not matter). The next circle will hold 12 individuals $(x_5, \ldots, x_{16})$, the individuals with the next best fitness values. The largest circle will have the less fit individuals from the population, and their number depends on the size of the population.

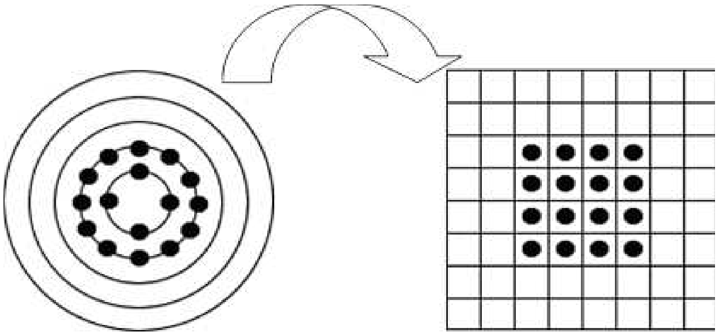


Figure 1. Concentric circles topology of the population and the corresponding two-dimensional grid topology

First of all, the individuals from the smallest circle (the fittest individuals of the population) will always be copied in the next population just as they are. This elitist choice is very suitable especially for algorithms that are using a relative fitness that is slightly different for each generation, because copying the best individuals in the next generation will mean that these individuals will be then tested again but using a different fitness function and they will survive only if they have a very good quality in this generation as well.

Each individual from the population will get the chance of being improved by involving it in a recombination process. The diversity will be thus increased, because considering each individual for recombination means to use genetic material of both very fit individuals and less fit individuals. The selection scheme will therefore decide the second parent involved in each recombination, and this is where the exploitation of the search space is pursued.

Therefore, for each individual except the best four that will be copied in the next generation, the selection scheme will choose its mate in the following way. Let us number the concentric circles on which the individuals are placed, so that the most exterior circle will have the value 0, and the most interior circle will have the biggest value. For a population size of n^2 (n even number), we will obtain $\frac{n}{2}$ circles, therefore the value $\frac{n}{2} - 1$ will be assigned to the most interior circle. For one individual belonging to circle $i, i = 0, \frac{n}{2} - 2$, we will always choose a mate from the circle $i + 1$. Because of the way individuals are placed on the circles, according to their fitness, this means that each individual from the population will be recombined with a better individual, but still close to it regarding the fitness value. This means that individuals from the smallest circle, even if they are not directly involved in recombination, will be chosen as mates for the individuals belonging to circle $\frac{n}{2} - 2$. We therefore have a so-called local selection that refers to the fact that individuals are selected only from a certain circle.

The local selection is done by using one of the existing selection operators like proportional selection, tournament selection and so on. A tournament selection scheme (Dumitrescu, 2000) is considered for all the experiments performed in this paper.

Once we have selected a pair of individuals, they will be recombined by using an existing recombination scheme, depending on their encoding (Back, 1997).

2.2. **Time Dimension.** The entire process described before takes place asynchronously, which is another distinctive and strong feature of the proposed search scheme. Both selection and recombination are done asynchronously. First, individuals from the circle $\frac{n}{2} - 2$ are considered for recombination. For each of them, an individual from the circle $\frac{n}{2} - 1$ is chosen according to a local selection and the two individuals will be recombined. The best offspring obtained after recombination will be mutated and the resulting individual will be accepted only if it has a better quality. The offspring, mutated or not, will then replace the first parent if it has a better quality. The elitist scheme that allows only better individuals to replace the first parents is counteracted by the fact that all individuals from the population are involved in recombination.

From the improved individuals of the circle $\frac{n}{2} - 2$ we will then choose mates for individuals belonging to the circle $\frac{n}{2} - 3$, according to the same local selection scheme. The process that results from the described scheme is a propagation process where the good genetic material of the fittest individuals will be first transferred to the closest fit individuals, and they will transfer it, together with their good genetic material, to the next fit individuals, and so on, from close to close, until the good genetic material collected from the entire population will reach the less fit individuals from the population.

2.3. **Circular Evolutionary Algorithm.** The algorithm that results from the proposed search scheme is called Circular Evolutionary Algorithm and is described in what follows.

```
Circular Evolutionary Algorithm
begin
    $t := 0$
    Initialize $P(t)$
    while (not stop-criteria) do
    begin
        Evaluate $P(t)$
        CircularSort $P(t)$
```

```
            for each circle $c$ ($c := \frac{n}{2} - 2, 0$)
            begin
                for each individual $i$ from $c$
                begin
                    $j$ :=LocalSelection(c-1)
                    $k$ :=Recombination $(i, j)$
                    Mutation($k$)
                end
            end
            $t := t + 1$
        end
    end
```

3. Evolving Network Topologies for Cellular Automata

The density-classification task is a prototypical distributed computational task for CAs defined as follows. We denote by ρ_0 the fraction (the density) of 1s in the initial configuration. The task requires deciding whether $\rho_0 > \frac{1}{2}$. If so, then the CA must go to a fixed-point configuration of 1s, otherwise it must go to a fixed-point configuration of 0s. The lattice size is chosen to be odd in order to avoid the case $\rho_0 = \frac{1}{2}$. Because finding the density of the initial configuration is a global task, and CA only relies on local interactions, this task is not trivial.

Due to the similarities between the ring lattice where each cell is linked to its r neighbors on each side and a graph where each node is connected to a limited number of nodes, even if not in the topological neighborhood, Watts proposed the use of a small-world graph instead of a ring lattice for CAs (Watts, 1999). He computed the performance of hand-constructed small-world graphs for the density task, and he obtained performance values bigger than 0.8, while the best performances of a cellular automaton based on a ring lattice topology were around 0.76. In order to obtain a different topology, he fixed the rule to a majority rule which states that a node will receive the state of the majority of its neighbors in the graph. Therefore, the problem that arises from Watts proposal is to evolve small-world networks topologies for the density task of CAs.

Besides the hand-constructed small-world networks proposed in (Watts, 1999), an evolutionary technique for evolving small-world networks for the density task has been proposed in (Tomassini, 2005). The authors used a cellular evolutionary algorithm (Alba, 2002) and they obtained topologies with a performance around 0.8, similar to the hand-constructed small-world networks of Watts. Moreover, this performance was obtained in most of the runs of the algorithm, while a good performance of ring lattice topology is difficult to obtain. When evolving small-world networks, they have started both from regular lattices and from random networks, and have studied the results obtained for both cases.

4. Detecting network configuration for density task using CEA

The proposed circular search model is applied for evolving network topologies for cellular automata, for the density task. The resulting algorithm is called Circular Evolution of Network Topologies (CENTA) and is described in what follows.

Encoding and Population Model

A potential solution of the problem represents an undirected graph describing the network topology. A two-dimensional grid is used to encode it. The fixed number of individuals from a population are distributed over the two-dimensional square grid. An array of integers represents all the nodes of the graph, and for each node we have an array of nodes connected to it.

The initial population consists of randomly generated regular lattices of size N =149, with a radius of 3, meaning that each node is connected to 3 nodes on both sides. One node in a graph can have a maximum of max connections, max being a parameter of the algorithm. The set of initial configurations is generated anew for each generation of the algorithm.

Fitness Assignment

The fitness function is a real-valued function $f : X \to [0, 1]$, where X denotes the search space of the problem. $f(x)$ represents the fraction of correct classifications over 100 initial configurations randomly generated but with a uniformly distributed density (Das, 1994).

Selection Operator

For each individual belonging to circle $i(i = 0, \frac{n}{2} - 2)$ a mate will be chosen from the circle $i + 1$. Because of the way individuals are placed on the circles, according to their fitness, this means that each individual from the population will be recombined with a better individual, but still close to it regarding the fitness value. The local selection used for choosing a mate from the circle $i + 1$ is a tournament scheme with a tournament size of $2(n - 2i - 1)$, where $4(n - 2i - 1)$ represents the number of individuals that belong to the circle $i(i = 0, \frac{n}{2} - 2)$. The selection for recombination is performed asynchronously, starting with the individuals belonging to circle $\frac{n}{2} - 2$ and continuing until we select mates for the individuals belonging to circle 0.

Recombination Operator

Once we have selected two individuals for recombination, a two-point crossover is used for our experiments. We start with the recombination of the fittest individuals from the population, thus giving them the opportunity to improve their fitness before they will be recombined with less fit individuals.

Mutation Operator

The individual resulted after each recombination will be mutated similar to the mutation proposed in (Tomassini, 2005), only that they consider a different scheme of choosing the individuals that will be subject of mutation. Each node of a graph that represents a possible solution for the problem will be mutated with a certain probability, parameter of the algorithm. For a node chosen for mutation we will either add or remove a link to another randomly chosen node, with a given probability.

Selection for Replacement and Survival

The replacement of the first parent with the best offspring obtained after recombination and mutation takes place asynchronously, due to the asynchronous selection and recombination scheme. The offspring will replace the first parent only if it has a better fitness.

The circular evolutionary algorithm is applied for evolving network topologies for CAs, for the density task. The parameters of the algorithm are written in Table 1.

Table 1. CENTA algorithm parameters

Population size	**100**
Probability of mutation	**0.5**
Max	**30**
Probability of adding a new link to node	**0.5**
Number of generations	**100**

The algorithm successfully evolves, in most of the runs, networks with performances around 0.8 for the density task. These results confirm the hypothesis of Watts regarding the fact that network topologies seem to be a better environment for local interactions that lead to a global behavior for the density task. On the other hand, the results could be interpreted as an indicator for the efficiency of the new proposed evolutionary technique.

Future work will investigate several static structural properties of obtained networks, such as degree distribution, clustering coefficient and average path length. The results will indicate the nature of evolved networks.

5. Conclusions

A new evolutionary search model has been proposed in this paper. The main features of the proposed model are a new population topology, which is distributed over concentric circles, according to the fitness of the individuals and an asynchronous selection and recombination of the individuals, which allows involving in recombination individuals that improve their quality, their adaptation to the environment from close to close.

The algorithm is applied for evolving network topologies for cellular automata, for the density task. The results obtained have been compared with the results reported by the authors of other techniques for these problems, and they can be considered as a proof for the efficiency of the proposed circular evolutionary model. The study of obtained networks will be the subject of a future paper.

References

[1] Alba E., Giacobini M., Tomassini M., Romero S. , *Comparing Synchronous and Asynchronous Cellular Genetic Algorithms*, J.J. Merelo et al. (eds.), Proceedings of the Parallel Problem Solving from Nature VII, Granada (SP), 2002, LNCS 2439, p. 601-610.

[2] Back, T., Fogel, D.B., Michalewicz, Z. (Ed.), *Handbook of Evolutionary Computation*, 1997, Institute of Physics Publishing, Bristol and Oxford University Press, New York.

[3] Das, R., Mitchell, M., Crutchfield, J. P., *A genetic algorithm discovers particle-based computation in cellular automata*, Parallel Problem Solving from Nature Conference (PPSN-III), 1994, Berlin, Germany, Springer-Verlag, p. 244-253.

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

[5] Tomassini M., Giacobini M., Darabos Ch., *Evolution and Dynamics of Small-World Cellular Automata*, Complex Systems, 2005, 15(4), p. 261-284.

[6] Watts, D., *Small worlds: The Dynamics of Networks between Order and Randomness*, Princeton University Press, Princeton, NJ, 1999.

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

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

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LIII**, Number 1, 2008

VIRTUAL ORGANIZATIONS – CONCEPTUAL MODELLING

DUMITRU RĂDOIU

Abstract. Although there is a vast literature on virtual organizations (VO), there's little agreement on this subject from the definition to the reference model. Explanation seems to be the lack of a consistent, easy to operate with and largely accepted definition and the absence of a framework to build useful conceptual models, based on formal theories. The present paper, presents an axiomatic, easy to operate with VO definition, the outline of VO concept modelling and a theorem supporting the idea that a consistent VO conceptual model must be backed up by a non-trivial formal theory (T). Empirically derived models – based on interpreting and generalising available "experimental" data - might lead to valid VO models but in the end, formalism (i.e. just formal description) cannot substitute formal theory.

Keywords: Virtual organization, conceptual model, formalism, formal theory

1. VO conceptual models

A VO conceptual model is a theoretical construct, an idealized logical framework, representing VO through a set of logical and mathematical objects (e.g. axioms, formulas, functions, processes). A conceptual model is explicitly chosen to be independent of implementation details. The value of a solid conceptual model is in the easiness of reasoning about VOs making it an important component of any scientific research. Reasoning on VOs with models is determined by a set of logical principles and regards a large spectre of aspects, starting from logically describing operation functions to theoretically evaluating hypotheses, and devising experimental procedures to test them (including computer simulations).

Developing a conceptual model within any domain, assumes the existence of domain ontology. By ontology we understand a consistent set of concepts within the domain and the relationships between those concepts. In the beginning of the paper we will address some general aspects regarding VO ontology.

Received by the editors: May 15, 2008.

2000 *Mathematics Subject Classification.* 68N30.

1998 *CR Categories and Descriptors.* K.6.3 [**Management of Computing and Information Systems**]: Software Management – *Software process.*

The description of a conceptual model assumes a description/modelling language. The paper identifies several required qualities of the modelling languages, necessary to develop VO conceptual models: syntactic quality (correctness, validity, completeness with respect to the VO domain) and semantic quality (validity and completeness). Validity regards correctness of mappings with regard to VO. Completeness regards relevancy of mappings in solving the problem. The above conclusions are based on an axiomatic definition of VO.

2. An axiomatic definition of VO

The Virtual Organization (VO) concept was introduced by Mowshowitz (1986) and started to be popularized by Davidow and Malone (1992). In the following years a very large number of papers approached this topic presenting VO as a way to address critical issues like accessing expertise in a global market, sharing skills and information within a network of independent entities which present themselves as a unified actor in a joint project, reducing costs, increasing flexibility, boosting innovation, accommodating increased demands from employees for a better quality of life.

A major effort to clarify the concept was conducted by Kasper-Fuehrer (2004) who reviewed a vast literature dealing with VO, written by German and English authors. Introducing an ideal definition Kasper-Fuehrer (2004) derives ten corollaries (natural consequences) which give the definition more accuracy and flexibility, in the same time dramatically diminishing its operability (i.e. easiness to operate with the definition in a formal manner).

We believe that an axiomatic definition of VO is simpler and eliminates the necessity of corollaries. Here are the axioms.

Axiom 1. *All co-operation forms (collective actions, following collective goals, based on dependencies among members) are enabled (in some instances created) by one or more forms of symbolic communication.*

The evidence of this axiom is based on the process of converting an unsecure transaction into a secure collaboration (well known aspect from the game theory). Symbolic communication is required to build the "trust mechanism", to convert the "unsecure game" into an "insurance game":

Axiom 2. *New forms of cooperation - enabled by new communication technologies – are successful only if they succeed to create new forms of wealth or add new value (e.g. info, sharing knowledge).*

Axiom 3. *The new forms of collective organizations may act to influence / further-develop the communication technology to secure their success.*

The reason for introducing Axioms 2 and 3 is evident: the success of the collective approach/action. If we accept these Axioms:

- We cast a clear light on the beginnings of collective actions (from individual action towards collective action involving complex dependencies)
- We underline the effect of the information and communication technologies (ICT) in the emergence of new forms of collective action
- Anticipate the emergence of new forms of organization as an effect of new communication technologies

In order to introduce the definition of VO, we start from one of the most largely accepted definition of an organization:

Definition 1. *An **Organization (O)** is a collective action among a group of members which pursues specific collective goals, controls its performance, and has a boundary separating it from its environment.*

Definition 2. *A **Virtual Organization (VO)** is an organization with the following characteristics:*

- *spatial: operates within geographically distributed work environments*
- *temporal: has a limited life span, until it performs its tasks or actions*
- *configurational: uses information and communication technology ICT to enable, maintain and sustain member relationships*

As the term organization is used in multiple ways, it is necessary to specify that in this paper we consider only **process-related type of organizations** (organization as task or action) as opposed to **functional organizations** (organizations as permanent structures) or **institutional organizations** (as an actual purposeful structure within a social context)

Examples:

(1) Co-operation with the goal of creating new forms of wealth is VOs using off-shore services in software development (dedicated individuals and/or teams belonging to different permanent organizations participate in a project)

(2) Co-operation with the goal of creating a new form of value is Open Source movement. It represents a new form of production (peer to peer production). E.g. Linux is world class software developed in a virtual organization.

And here are a few observations:

- The VOs set is a subset of the Os set
- Spatial distribution condition is critical for is the reason for the manifestation of virtuality within a VO
- By definition, all VOs are temporary and only exist until they perform towards their collective goals, be it the creation of (new form of) wealth or (a new) value

- The definition can accommodate new forms of VOs as new forms of symbolic communication emerge (e.g. avatar face-to-face communication in virtual worlds like Second Life)

3. Some ontology aspects of VOs

As we mentioned before, conceptual modelling assumes the existence of a consistent set of concepts within the VO domain and the relationships between those concepts. By VO ontology we understand the common vocabulary describing the concepts, the actors and relationships.

We start with a few observations on architecture. For a significant number of VOs the Internet is the network supporting ICT services for:

- Creating, gathering, integrating and distributing information throughout the organization components
- Sharing (human) resources via platforms through which individuals collaborate.

Noting that technology is just the platform over which information is distributed and resources shared, we can consider three large areas which should addressed in VO ontology and concept modelling:

- VO Enabling Technology (the platform, e.g. applications, components), enabling information to be created, distributed, consumed (e.g. interoperability, standardization, security, speed)
- VO Processes Design, or how VO uses the platform (how we create, organize, distribute and consume information, how it supports the VO operations (procedures), and tactics (business processes)
- VO Superstructure (business level), or how we create new wealth and/or new value (strategy, trust mechanisms, organization)

In Figure 1 a simplified **business architectures** is represented.

The role of each architecture level is to support the level above and in a first approximation ("week interaction approximation") one can develop a theory and an "independent" model for each level considering its own role is to support the level above. (The approximation consists in neglecting the changes/optimisations of any supporting level, induced by the level above). VO involves coupling at different levels and coupling assumes the existence of models.

Based on VO goals, strategy and organization, next step would be to select the most adequate **topology** of the type of VO we want to model. The main topologies are **process oriented, peer-to-peer** or **main contractor topology**. Topology clarifies not only at what levels the parties need to manifest interoperability but also their major roles in the VO.

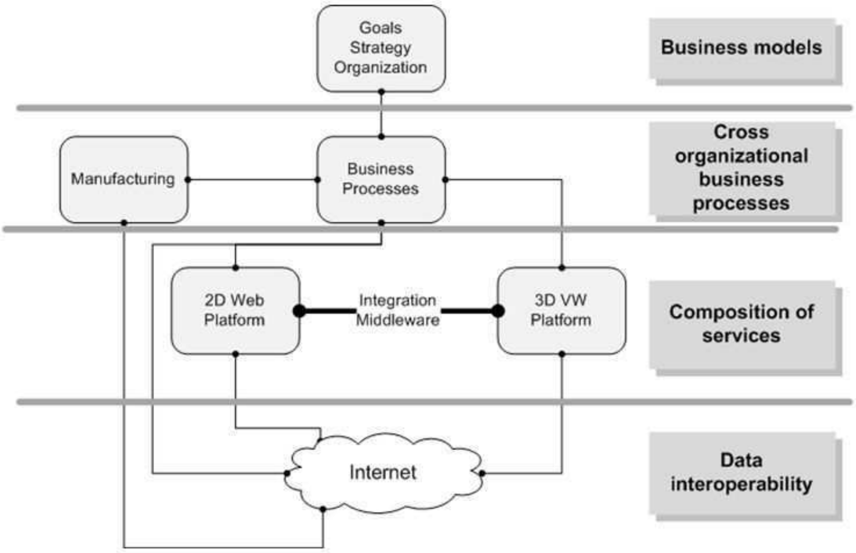


FIGURE 1. Basic VO architecture types: financial services, telecommunications, and manufacturing (adapted from Radoiu, 2008)

Based on topology, next step is to identify **VO life cycle functions**, and model them for each phase: **creation, operation, evolution, and dissolution**; emphasise on operation.

As any VO is based on **interoperability**, it follows that this also must be modelled and most likely at all levels: business level (e.g. organizational roles), business processes (e.g. sharing knowledge assets), enabling technology (e.g. communication components).

It appears obvious that **ontology descriptions** at all levels must be developed and used in order to secure consistent VO concept modelling.

It also appears evident that a "complete conceptual model" will actually consist of a collection of models developed at each architecture level, for a specific topology and for all VO lifecycle functions.

4. TOWARDS VO CONCEPTUAL MODELLING

Models – for each of the levels in discussion - are developed using a modelling environment. Modelling environments (platforms, languages, tools and paradigms) are specific to each level (as we have different semantic needs at each level, it is likely that we use different modelling languages at each architecture level). At this moment, the author is not aware of any concept model of

VOs. The explanation seems to be the substitution of the formal theory with formalism and it is here where we would like to make some comments. Comments refer to the modelling process which makes all the difference between formalism and formal modelling. The difference between the two concepts (formalism vs formal theory in VO modelling) has been firstly discussed by Putnik, (2004).

The main elements of every **modelling environment** are: the **modelling language (L)** (containing the elements with which a model can be described) and the **modelling algorithms** for each step of the modelling process which validate the outcome of the **modelling process**.

Advantages of using a **modelling process** (and we underline the word process) are:

- One can predict the outcome (e.g. a valid and successful conceptual model)
- The outcome of the modelling process (the conceptual model) depends on the capability of the modelling process
- An algorithmic process enables consistent quantitative approaches (use of metrics)
- An algorithmic modelling process lowers the risks with regard to the value of the VO model

For virtual organizations, obviously, the modelling environment must permit the modelling of all characteristics of a VO (e.g. spatial, temporal and configurational) plus the modelling of the coupling mechanisms among parties in a VO (having in mind the topology) and most important, the VO life cycle functions.

In order to reveal the requirements of VO conceptual modelling we will use the analogy between mathematical abstract models and VO concept models.

A **mathematical model** is an **abstract model** that uses **mathematical language** (**formal language**) to describe an existing system (or a system to be constructed) which presents knowledge of that system in usable form (Sakharov, 2008).

A **formal language** is defined by an alphabet and formation rules. Syntactically complex languages are defined by means of **grammars** or regular expressions (Sakharov, 2008). The alphabet is a set of symbols used to build the language. The formation rules specify which strings of symbols count as well-formed.The **well-formed strings** of symbols are also called **words, expressions, formulas, or terms**. The **formation rules** are usually recursive. Some rules postulate that such and such expressions belong to the language in question. Some other rules establish how to build well-formed expressions from other expressions belonging to the language.

Any meaningful mathematical model is based on a theory, the **theory** being the set of sentences which is closed under logical implication. One of its useful aspects is that, given any subset of sentences $\{s_1, s_2, ...,\}$ in the theory, if sentence r is a logical consequence of $\{s_1, s_2, ...,\}$, then r must also be in the theory (Weisstein, E. W.).

Back to VO conceptual modelling, let us define a **VO conceptual model** as an **abstract model** that uses **modelling language** to describe the system. The modelling language is described by its **alphabet, syntax (grammar)**, *and* **semantics**. The similarity with mathematical modelling is straightforward. Any useful VO conceptual model must be described by a **theory**. A set of sentences closed under a logical implication. But, there's at least one more aspect.

In connection with the modelling language we have a **syntax constructs domain** (D), defined as the set of all formulas which are part of the language. We note $\mathbf{L}^S$ the **modelling language** built with the syntax S over a given alphabet. Worth mentioning here that there are two major approaches to describe a syntax: **graph grammars** and **meta-models** (i.e. a meta-model is the model of the syntax) (Karagiannis, 2002). As such, a language is useless as long as we do not associate meanings to all its syntax constructs (terms, formulas, sentences). The meaning is given by a chosen set of "meaning values" (i.e. significance values) which are formally named the **semantics domain** (V). It follows than that the **semantics** of a **language** $\mathbf{L}^S$ describes the significance of a modelling language by mapping (say a map a) the set of all syntax constructs (D) to a set of significance values (V). Obviously, in our case, the significance values must be related to the VO theory. The map a will allocate a VO-related meaning to every syntactically correct construction in a given language. Because of the degree of freedom in choosing the set (V), different syntax constructs, expressed in different modelling languages, could have the same meaning/interpretation. It follows that there's not only one language susceptible to describe VOs.

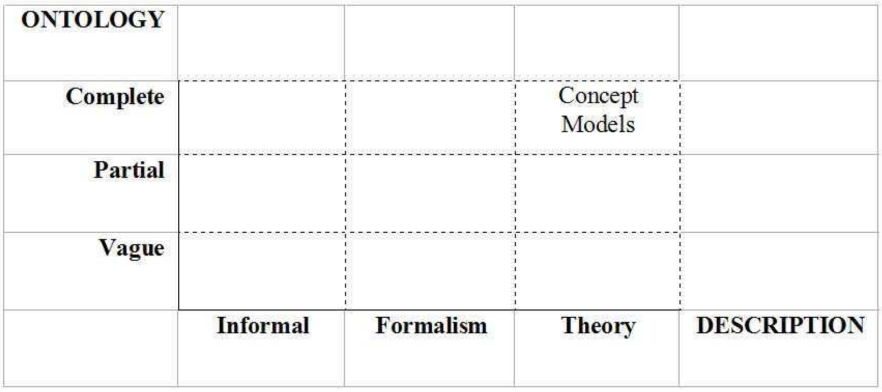
The map a and the domain (V) form a **structure** and is noted *VOstructure* $= (V, a))$;

If under a given **interpretation** a **formula** becomes true, then that **interpretation** is a **model** of that **formula**. A **sentence** is called **satisfiable** if there is at least one **interpretation** under which that sentence is true.

When applying an **interpretation** to a **sentence** the assignment of variables is irrelevant (as the **sentence** has no variables occurring free). Thus one can say that a **sentence** is **satisfiable** if exists at least one **structure** making the **sentence** true, that is, if exists at least one **structure** that is a **model** of the **sentence**. As seen before, some of the **formulas** that constitute a **language** are **sentences**. From all those **sentences** some will eventually be **satisfiable**. The subset of all **sentences,** whose elements are **satisfiable sentences**, **and closed under consequence**, is a **Theory**. Because **sentences** are **satisfied** by *VOstructure,* the referred subset of **sentences** can be designated by **Theory of *VOs***.

Theorem 1. *The condition of a given modelling language (L^S) to correctly describe a VO conceptual model is to permit the description of at least one nontrivial theory (T) on that VO.*

And here is an informal proof of this theorem. If the sentences describing the model do not verify the axioms and are not consistent with the definitions, than the model is not describing a VO. Let us assume that the axioms are verified. Another group of sentences must completely describe the architecture, topology, operation functions, and interoperability functions. We find ourselves in the situation where the sentences are satisfied by a VO*structure.* As the modelling sentences have to be true under the same restrictions, these sentences can be referred as the theory of VO. Obviously, not any modelling language (L^S) fulfils the above criteria. Next to general modelling language qualities (syntactic correctness and validity), a few other are essential: syntactic completeness and semantic validity and completeness with respect to the VO domain (i.e. to be able to describe the theory behind VO). Semantic validity regards correctness of mappings with regard to VO. Completeness regards relevancy of mappings in solving the conceptual modelling problem.



ONTOLOGY				
Complete			Concept Models	
Partial				
Vague				
	Informal	Formalism	Theory	DESCRIPTION

TABLE 1. Models classification with regard to description and ontology

Another observation is that semantic completeness requires a complete ontology of the domain.

In the end, it is fair to ask what the benefits of VO concept modelling are. After all, working VO models were discovered and could be perfected through observations (e.g. analyzing and generalizing empirical data, case studies).

The immediate risk is that a theoretical approach alone is likely to produce results without practical importance. The utility of a theoretical approach is several folds. Firstly, a formal theory approach always proves to have utility for "engineering" tasks such as design and implementation of VOs, because it provides the "desired efficiency and effectiveness" (Putnik 2005). Secondly, it could be used to explain the success or failure of different VOs, their emergence or absence in the market. Thirdly, it could help existing VOs to lower the risk in their future development and help them better understand the mechanisms of their informally and iteratively developed working model. And it could also help future VOs in developing their working model. Obviously, the conditions for VO emergence exist for some time now, but only those which were able to develop (so far, empirically) a consistent working model, survived.

5. Comments around a simple example and conclusions

Let us take a very simple example: a VO consisting of two parties, dedicated to custom software development. The theory which must be supported by modelling language (e.g. the logic of an operation function at the business process level) will be described at the highest level by the meta-model which concepts and operations must be reflected by the appropriate modelling language. The spatial distribution of VO allow us to focus only at the business process level and ICT services level assuming that ICT infrastructure is the Internet. At the business level, the goal is developing custom made software applications, the strategy is to address the market needs using geographically dispersed resources, the organization consists of high level design process undertaken by one party, low level design, development, and testing with the other. Let us assume that the trust mechanism is contractual. The topology is obviously of the type "main contractor". Starting from the theory describing the methodology of custom software development process, and introducing the constraints (the process is being carried out by a distributed organization), the modelling language for the business process level must be able to consistently describe business processes interoperability: input, output, coordination events and events notification). The modelling language must be able to model interoperability at the ITC services level (e.g. files synchronisation). The VO conceptual model will consist of several models:

- Operational model (formally describing the realization of the operations functions)
- Evolution model (formally describing VO growth)
- Integration models at all three levels: business level, business processes, enabling technology (formally describing how parties will practically integrate)

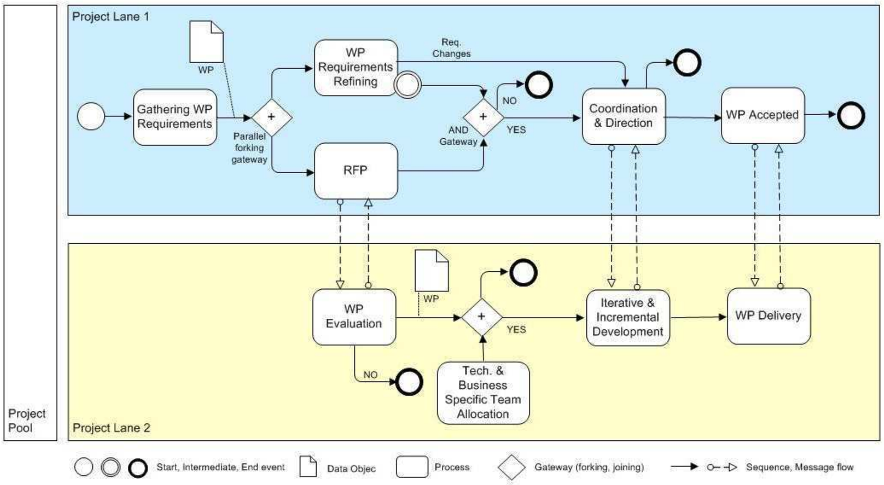


FIGURE 2. One operation function (software development), modelled with BPMN, for business processes level

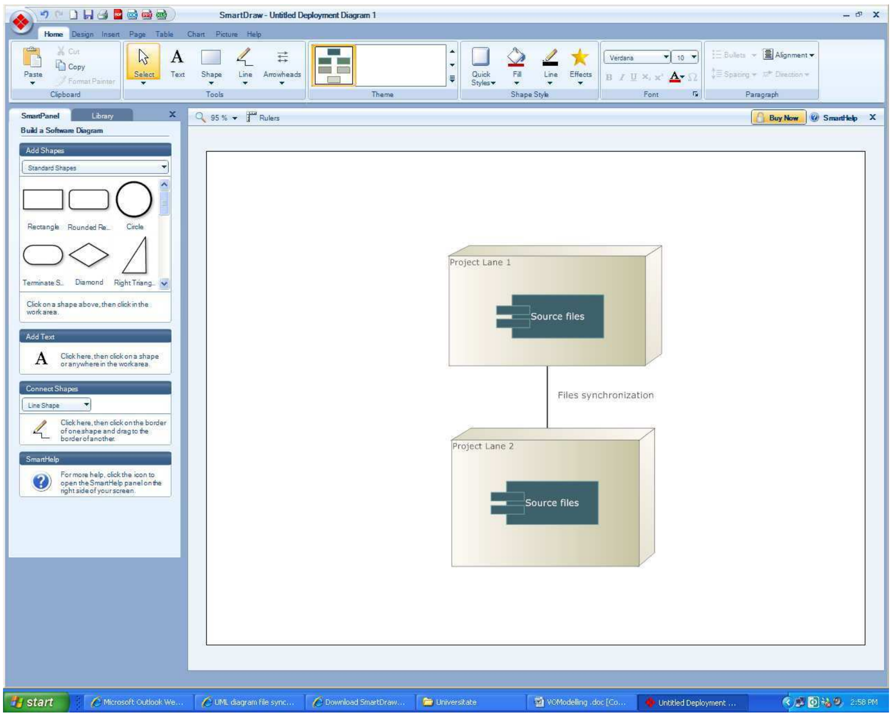


FIGURE 3. One integration function (file synchronization), modelled with UML, for two parties VO ITC services level

Two of these models are depicted in Fig. 2 and 3.

Let Σ be the set of sub-processes necessary to cover the custom software development workflow in the VO, in line with the specifications of such a workflow. Let be Σ_1 the sub-processes taking place within Lane 1 and $\Sigma_2$the sub-processes taking place within Lane 2.

$\Sigma = \Sigma_1$U Σ_2 (set of sub-processes necessary to cover the custom software development workflow is the reunion of Σ_1 and Σ_2. AS All of these sub-processes are mandatory, the interfaces between the two project lanes must formally describe the channels for input data, output data, event notification, and monitoring and control events.

Today "VO research area is recognized as a scientific discipline" and it is probable that "in 2015 most enterprises will be part of some sustainable collaborative networks that will act as breeding environments for the formation of dynamic virtual organizations in response to fast changing market conditions", Camarinha (2002, 2003). The drivers for this trend (Gartner, 2007) are: maturing technology and standards (e.g. EDI, XML, GDS, CPFR, SOA, Web services, virtualization, semantic Web (Berners-Lee, 2003)), improved integration (integration as a service) and potential business impact.

Conceptual modelling proves its utility in the design and implementation of VOs also helping existing VOs to lower the risk in their present operation and future development.

References

Berners-Lee, T: Semantic Web Road map. W3C Design Issues, http://www.w3.org/DesignIssues/Semantic.html, last change 2003/01/06

Camarinha-Matos L-M, Afsarmanesh H: A Roadmap for Strategic Research on Virtual Organizations, in "Processes and foundations for virtual organizations", (L.M. Camarinha-Matos, H. Afsarmanesh, Editors), Kluwer Academic Publishers, ISBN 1-4020-7638-X, Oct 2003

Camarinha-Matos L-M, Afsarmanesh H: THINKcreative Interim Green Report on New Collaborative forms and their needs, THINKcreative report, (L.M. Camarinha-Matos, H. Afsarmanesh, Editors), Sep 2002

Davidow, W., and M. Malone: 1992. The Virtual Corporation: Structuring and Revitalizing the Corporation for the 21st Century. New York: Harper Collins.

Eva C. Kasper-Fuehrer and Ashkanasy, Neal M.: Int. Studies of Mgt. & Org., vol. 33, no. 4, Winter 2003–4, pp. 34–64.

The Gartner Scenario: Current Trends and Future Direction of the IT Industry, April 22-26, 2007, San Francisco, CA

Radoiu D. (2008), Virtual Organizations in emerging3D Virtual Worlds, in submitted to 14th International Conference on Concurrent Enterprising – ICE2008, Lisbon, Portugal

Mowshowitz, A. 1986. "Social Dimensions of Office Automation." In Advances in Computers, vol. 25, ed. M. Yovitz, 335–404. New York: Academic Press.

Putnik, G. and Sousa, R. (2004). On Formal Theories for Virtual Enterprises, in submitted to PRO-VE'04 5th IFIP Working Conference on Virtual Enterprises, Toulouse, France.

Putnik, G. D., Cunha, M. M., Sousa, R., Ávila, P. (2005) BM_ Virtual Enterprise: A Model for Dynamics and Virtuality, in Putnik G., Cunha M. M. (Eds.) Virtual Enterprise Integration: Technological and Organizational Perspectives, IDEA Group Publishing, Hershey, PA, USA, pp: 124-143

Turner, K. J., Ed. (1993). Using Formal Description Techniques: An Introduction to ESTELLE, LOTOS and SDL, John Wiley & Sons.

Sakharov, A.. "Formal Language." From MathWorld–A Wolfram Web Resource, created by Eric W. Weisstein. http://mathworld.wolfram.com/ FormalLanguage.html (2008/20/03)

Weisstein, E. W. "Theory." From MathWorld–A Wolfram Web Resource. http://mathworld.wolfram.com/Theory.html (2008/20/03)

Petru Maior University, 1 Nicolae Iorga St., Târgu Mureş, Romania
E-mail address: Dumitru.Radoiu@Sysgenic.com

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LIII**, Number 1, 2008

IMPROVING THE ACCURACY OF DIGITAL TERRAIN MODELS

GABRIELA DROJ

Abstract. The change from paper maps to GIS, in various kinds of geographical data analysis and applications, has made it easy to use the same spatial data for different applications and also for combining several layers into quite complex spatial models, including the three-dimensional reference space (surface), known as Digital Terrain Model (DTM). The objectives of this study are: (1) to compare the different algorithms involved in producing a DTM, (2) to establish the factors which affect the accuracy of the DTM and (3) to improve the quality of the generated DTM.

1. Introduction

The change from paper maps to digital data, in various kinds of geographical data analysis and applications, has made easy to use the same spatial data for different applications and also for combining several layers into quite complex spatial models.

Using GIS technology, contemporary maps have taken radical new forms of display beyond the usual 2D planimetric paper map. Today, it is expected to be able to drape spatial information on a 3D view of the terrain. The 3D view of the terrain is called Digital Terrain Model (DTM) or Digital Elevation Model (DEM) [2].

The digital terrain models are frequently used to take important decisions like to answer hydrological questions concerning flooding. In order to judge these decisions the quality of DTM must be known. The quality of DTM is, unfortunately, rarely checked. While the development of GIS advances, DTM research has so far been neglected. The objectives of this study are: (1) to compare the different algorithms involved in producing a DTM, (2) to establish the factors which affect the accuracy of the DTM and (3) to improve the quality of the generated DTM.

Received by the editors: June 10, 2007.

2000 *Mathematics Subject Classification.* 65D05, 65D07, 68U05.

1998 *CR Categories and Descriptors.* I.3.5 [**Computational Geometry and Object Modeling**]: Subtopic – *Curve, surface, solid, and object representations* I.3.7 [**Three-Dimensional Graphics and Realism**]: Subtopic – *Visible line/surface algorithms*.

2. Digital Terrain Modeling

A digital terrain model (DTM) is a digital representation of ground surface, topography or terrain. It is also known as digital elevation model (DEM). DTM can be represented as a raster (a grid of squares), as contour lines or as a triangular irregular network (TIN) [3, 4, 9].

Two methods are frequently used to obtaining digital terrain model: cartographic digitizing, or automatic measurements [6]. The cartographic digitizing method is widely used because topographic maps are usually available. The input data form the basis for the computation of the DTM, consisting of points. The computation itself consists in spatial interpolation algorithms. Automatic measurement techniques like photogrammetry and airborne laser scanning, output bulk points, with a high density. The DTM is realized in the post processing phase usually by creating the TIN or by interpolation.

2.1. Interpolation.

The current research and industrial projects in GIS require higher standards for the accuracy of spatial data. The data in geographical informational systems (GIS) are usually collected as points, where a point is considered a triplet (x,y,z), where x and y are the coordinates of the point and z is the specific information. This specific information can be for example: the altitude level in the point (x, y), the quantity of precipitations, the level of pollution, type of soil, socio-economic parameters etc.. The mapping and spatial analysis often requires converting this type of field measurements into continuous space. Interpolation is one of the frequently used methods to transform field data, especially the point samples into a continuous form such as grid or raster data formats.

There are several interpolation methods frequently used in GIS. The following eight widely used methods are compared and studied in this paper. These methods are Inverse distance weighted (IDW), Spline Biquadratic interpolation, Spline Bicubic interpolation, B-spline interpolation, Nearest neighbors - Voronoi diagrams , Delaunay Triangulation, Quadratic Shepard interpolation, Kriging interpolation [1, 5, 7].

2.2. Quality of surface.

Measurement of errors for the results is often impossible because the true value for every geographic feature or phenomenon represented in this geographic data set is rarely determinable. Therefore the uncertainty, instead of error, should be used to describe the quality of an interpolation result. Quantifying uncertainty in these cases requires comparison of the known data with the created surface [10].

To analyze the pattern of deviation between two sets of elevation data, conventional ways are to yield statistical expressions of the accuracy, such as the root mean square error, standard deviation, mean, variance, coefficient of variation. In fact, all statistical measures that are effective for describing a frequency distribution, including central tendency and dispersion measures, may be used, as long as various assumptions for specific methods are satisfied [6, 10].

For the evaluation of DTM the most widely used measure, usually the only one, is the well known Root Mean Square Error (RMSE). Actually, it measures the dispersion of the frequency distribution of deviations between the original elevation data and the DTM data, mathematically expressed as:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(z_{d,i} - z_{r,i})^2}$$

where: $Z_{d,i}$ is the ith elevation value measured on the DTM surface; $Z_{r,i}$ is the corresponding original elevation; n is the number of elevation points checked.

It's important to mention that for an accurate evaluation, is necessary to evaluate the surface by using an independent set of data. In this case the RMSE mirrors the quality of the surface but for a correct evaluation we recommend to analyze at least one more statistical index like standard deviation or Median absolute deviation.

3. Experiments with real-world data

DTMs are the most popular results of interpolation. In the following, we will test different methods and algorithms for creation of DTM in order to compare the different algorithms involved in producing a DTM, to establish the factors which affect the accuracy of the DTM, and to determine how to improve the quality of the generated DTM. To test and compare the methods with real data we have selected an area from north hills of Oradea municipality.

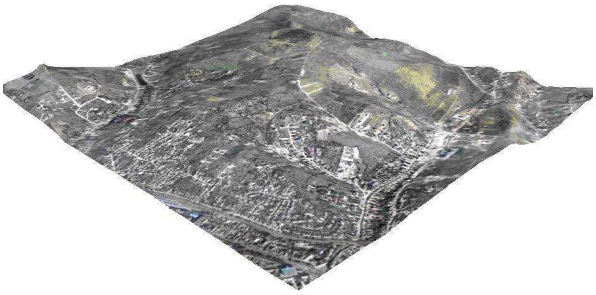


Figure 1. The Oradea 3D Model

For the first DTM we used photogrammetric measurement of spot elevations from orthorectified airborne image of the area. This is the fastest way of obtaining the digital elevation model. In the case of collecting data with close-range photogrammetry or airborne laser scan the result consists in a high density of points with three coordinates (x,y,z). By computing the TIN model of these points we obtain fast a DTM. The TIN of the area was generated using ARCGIS Desktop 9.1. In the figure 1 we represented the 3D model created.

This model will consider the reference for testing and evaluating the most popular interpolation algorithms used for DTM creation. These methods are Inverse distance weighted (IDW), Spline Biquadratic interpolation, Spline Bicubic interpolation, B-spline interpolation, Nearest neighbors - Voronoi diagrams, Delaunay Triangulation, Quadratic Shepard interpolation and Kriging interpolation.

The comparisons of these algorithms were made by analyzing their results. The evaluation of the created surfaces was made by direct observation: with visual comparisons of the models using the spot image as reference and by using statistical parameters.

The first step was to create a regular grid with the step of 500 m, for a total of 30 points. On this set of points we tested the algorithms specified before. In Figure2 we represented the results of these algorithms.

In the second experiment, we have created a regular grid with the step of 250 m, for a total of 121 points. On this sets of points we tested the algorithms specified before. In Figure 3 we represented the results of these algorithms.

The visual comparisons show a high similarity with the reference for the Delaunay Triangulation and Shepard interpolation in both test cases.

In order to evaluate the surfaces generated by using statistical methods it is necessary to test the quality of the surfaces with an independent set of data, which were not considered in the interpolation.

In the following we will evaluate the surfaces generated by using a random set of points for which was determined the real value of the altitude. For this independent random set we have determined the following statistical parameters: variation, median absolute deviation, standard deviation and the root mean square error. The determined values are presented in the table below:

4. Results of test cases

The results obtained in the both cases show that the most accurate surfaces are generated, for the first case (grid of 500 m), by Kriging, Shepard and B-spline algorithms and for the second case (grid of 250m) by Delauney triangulation followed by Shepard and Kriging.

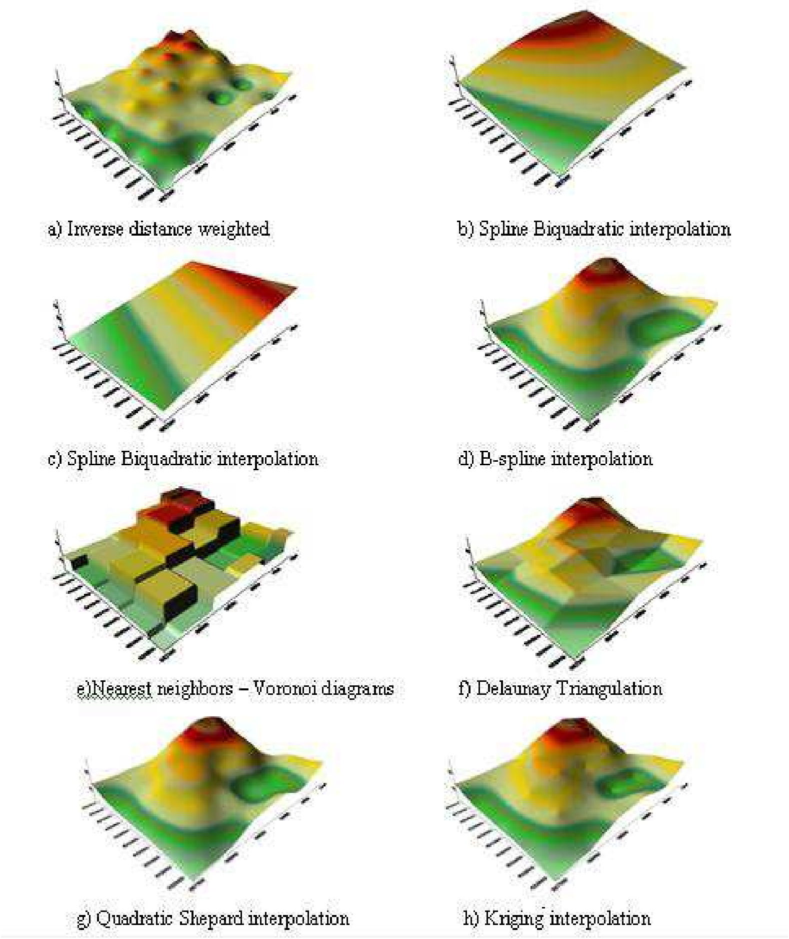

a) Inverse distance weighted
b) Spline Biquadratic interpolation
c) Spline Biquadratic interpolation
d) B-spline interpolation
e)Nearest neighbors – Voronoi diagrams
f) Delaunay Triangulation
g) Quadratic Shepard interpolation
h) Kriging interpolation

Figure 2. DTM model generated with 30 known points

If we evaluate all the statistical data we notice that the Delauney triangulation represents the optimal method. Similar results can be obtained by Kriging and Shepard interpolation. Even these methods are sometimes more efficient than the Delauney triangulation. Nevertheless, the Delauney algorithm is recommended because it needs less computing time and it is not changing the original values of the points. The b-Spline algorithm gives also a good result but in this case the computing time is much higher and it is smoothing the surface, fact which is making this method inadequate for surfaces with a high altitude difference.

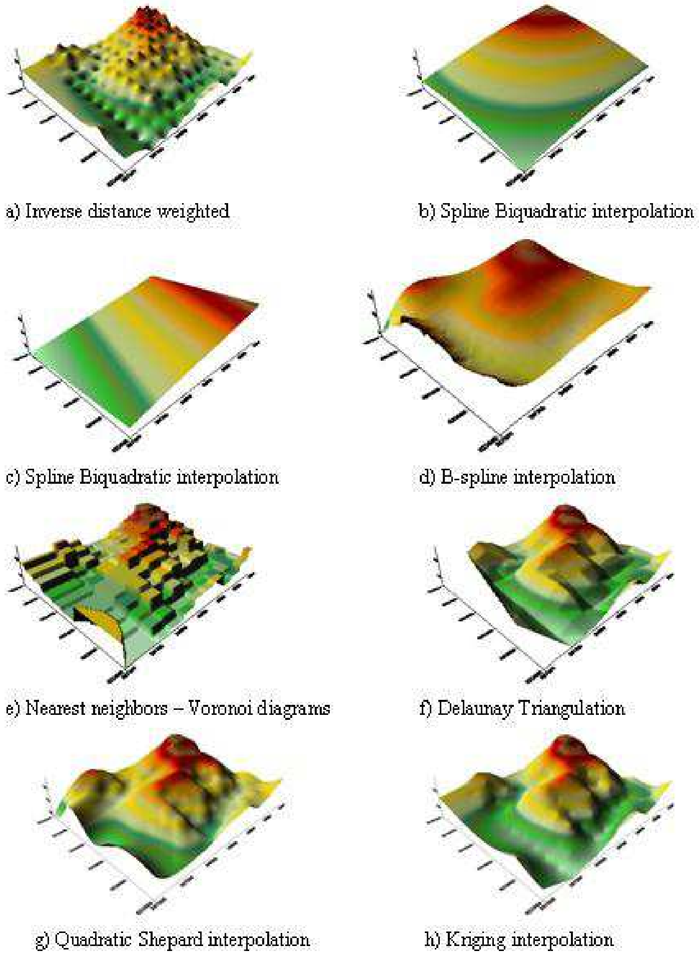

a) Inverse distance weighted
b) Spline Biquadratic interpolation
c) Spline Biquadratic interpolation
d) B-spline interpolation
e) Nearest neighbors – Voronoi diagrams
f) Delaunay Triangulation
g) Quadratic Shepard interpolation
h) Kriging interpolation

FIGURE 3. DTM model generated with 121 known points

Many studies have examined the DTMs generated by interpolation but the earlier studies compare fewer methods, usually by testing a strong algorithm with a fast one or by testing one algorithm with two or three different point density. Some examples of studies are: a comparison between IDW and Voronoi [4], IDW vs. Kriging [11], and between IDW, Kriging, Thiessen polygons and TIN [8] and IDW, minimum curvature, modified Shepard and TIN [10]. Except the study made by Weng, all of them are using only one set of input data.

Our results regarding the accuracy of surfaces created by interpolation are similar with the earlier studies. By testing a higher number of methods and in

Methods	Variation 250	Variation 500	Median absolute deviation 250	Median absolute deviation 500	Standard deviation 250	Standard deviation 500	RMSE 250	RMSE 500
IDW	231	167	11.6	10.17	15.21	12.29	13.48	10.93
Biquadratic	390	426	16.17	16.56	19.75	20.64	15.35	13.84
Bicubic	947	1044	26.53	26	30.77	32	19.14	17.14
B-spline	166.2	145	9.67	9.9	12.89	12	12.54	10.37
Voronoi	246	94.5	11.38	7.3	15.71	9.75	13.79	9.27
Delaunay	218	36.63	10.85	3.41	14.78	6.05	13.34	7.99
Shepard	164	71.78	9.56	6.40	12.81	8.47	11.97	8.49
Kriging	160.06	78.97	9.74	6.81	12.65	8.88	11.75	8.94

TABLE 1. The statistics for all the created surfaces,based on a set of random control points

two different cases we demonstrated that DTM accuracy can vary to a certain degree with different interpolation algorithms and interpolation parameters.

5. Conclusion

In this paper we compared the most popular algorithms involved in producing a DTM, in order to establish the main factors which affect the accuracy of the DTM and to improve the quality of the generated DTM.

The performance of eight methods, in two different cases, was evaluated in this paper, based on the accuracy of the generated surface. The first conclusion which is pointed up by the values presented in table 1 and by the analysis of the visual results, in the both cases, shows that input data form the basis for the computation of the DTM. The density of the known points is a more important factor in increasing DTM quality than the algorithm used in surface creation.

A second conclusion is that there is no optimal algorithm for any situation, the results given by different computational method are influenced by more factors like: the conformation of the field, the density of the initial points, the quality of the known values and nevertheless by algorithm used.

The last conclusion is that, we can improve the quality of the DTM by using the Delauney algorithm and a high density of known points. Regarding the way of computing for Delaunay triangulation, it is necessary to have at least 3 points for each hill or valley.

References

[1] M. R. Asim, M. Ghulam, K. Brodlie, *"Constrained Visualization of 2D Positive Data using Modified Quadratic Shepard Method,"* WSCG'2004, Plzen, Czech Republic.

[2] J. K. Berry and Associates *"The GIS Primer - An Introduction to Geographic Information Systems,"* http://www.innovativegis.com/basis/, May 2006.

[3] L. De Floriani, E. Puppo, P. Magillo *"Application of Computational Geometry to Geographic Informational Systems,"* Handbook of Computational Geometry, 1999 Elsevier Science, pp. 333-388.

[4] Chongjiang Du *"An Interpolation Method for Grid-Based Terrain Modeling,"* The Computer Journal,vol.39, nr. 10, 1996

[5] ESRI. Environmental Science Research Institute *"Arc/Info 8.0.1,"* ESRI, Inc. 1999.

[6] W. Karel, N. Pfeifer, C. Briese *" DTM Quality Assessment,"* ISPRS Technical Commission II Symposium , 2006 XXXVI/2, pp. 7-12.

[7] R. T. Trambitas *"Analiza numerica. Note de curs,* http://math.ubbcluj.ro/ tradu/narom.html, 2003

[8] P. Siska, I. Hung *"Assessment of Kriging Accuracy in the GIS Environment,"* Annual ESRI International User Conference, San Diego, CA, 2001.

[9] M. van Kreveld *" Algorithms for Triangulated Terrains,"* Conference on Current Trends in Theory and Practice of Informatics, 1997.

[10] Qihao Weng, *"Quantifying Uncertainty of Digital Elevation Model,"* http://isu.indstate.edu/qweng, December 2006.

[11] Y. Ziary, H. Safari, *"To Compare Two Interpolation Methods: IDW, KRIGING for Providing Properties (Area) Surface Interpolation Map Land Price,"* FIG Working Week, 2007

University of Oradea, Department of Geodesy and Topography
E-mail address: gaby@oradea.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LIII**, Number 1, 2008

MINIMUM FLOW ALGORITHMS. DYNAMIC TREE IMPLEMENTATIONS

ELEONOR CIUREA, OANA GEORGESCU, AND MIHAI IOLU

Abstract. We present augmenting path algorithms for minimum flows with dynamic tree implementations. The time bounds for augmenting path for minimum flows automatically improve when the algorithms are implemented with dynamic tree.

1. Introduction

The computation of a maximum flow in a graph has been an important and well studied problem, both in the fields of computer science and operations research. Many efficient algorithms have been developed to solve this problem, see, e.g., [1], [3]. Sleator and Tarjan [7] developed the dynamic tree data structure and used it to improve the worst-case complexity of Dinic's algorithm from $O(n^2m)$ to $O(nmlogn)$. Since then, researchers have used this data structure on many occasions to improve the performance of a range of network flow algorithms. Using the dynamic tree data structure, Goldberg and Tarjan [6] improved the complexity of the FIFO preflow-push algorithm from $O(n^3)$ to $O(nmlog(n^2/m))$, and Ahuja, Orlin and Tarjan [2] improved the complexity of the excess scalling algorithm and several of its variants.

The computation of a minimum flow in a network has been investigated by Ciurea and Ciupala [4]. The algorithms for minimum flows with dynamic tree implementations were not treated. In this paper we present the augmenting path algorithms for minimum flows with dynamic tree implementations.

In the presentation to follow, we assume familiarity with flow problems and we omit many details. The reader interested in further details is urged to consult the books [1, 3] for maximum flow problem and the paper [4] for minimum flow problem.

Received by the editors: May 1, 2008.

2000 *Mathematics Subject Classification.* 90B10, 90C35, 05C35, 68R10.

1998 *CR Categories and Descriptors.* code G.2.2 [**Graph Theory**]: Subtopic – *Network problems* .

2. Terminology and Preliminaries

We consider a capacitated network $G = (N, A, b, c, s, t)$ with a nonnegative capacity $c(x, y)$ and with a nonnegative lower bound $l(x, y)$ associated with each arc $(x, y) \in A$. We distinguish two special nodes in the network G: a source node s and a sink node t.

For a given pair of not necessarily disjoint subset X, Y of the nodes set N of a network we use the notation:

$$(X, Y) = \{(x, y)|(x, y) \in A, x \in X, y \in Y\}$$

and for a given function f on arcs set A we use the notation:

$$f(X, Y) = \sum_{(X,Y)} f(x, y)$$

A flow is a function $f : A \to R+$, satisfying the next conditions:

$$f(x, N) - f(N, x) = \begin{cases} v, & x = s \\ 0, & x \neq s, t \\ -v, & x = t \end{cases} \tag{2.1a}$$

$$l(x, y) \leq f(x, y) \leq c(x, y), \forall (x, y) \in A \tag{2.1b}$$

for some v. We refer to v as the value of the flow f. The maximum (minimum) flow problem is to determine a flow $\tilde{f}(\hat{f})$ for which v is maximized (minimized). A cut is a partition of the node set N into two subset S and $T = N - S$; we represent this cut using the notation $[S, T]$. We refer to a cut $[S, T]$ as a $s - t$ cut if $s \in S$ and $t \in T$. We refer to an arc (x, y) with $x \in S$ and $y \in T$ as a forward arc of the cut and an arc (x, y) with $x \in T$ and $y \in S$ as a backward arc of the cut. Let (S, T) denote the set of forward arcs in the cut and let (T, S) denote the backward arcs.

For the maximum flow problem we define the capacity $\tilde{c}[S, T]$ of a $s - t$ cut $[S, T]$ as:

$$\tilde{c}[S, T] = c(S, T) - l(T, S) \tag{2.2}$$

and for the minimum flow problem, we define the capacity $\hat{c}[S, T]$ of a $s - t$ cut [S,T] as:

$$\hat{c}[S, T] = l(S, T) - c(T, S) \tag{2.3}$$

We refer to a $s - t$ cut where capacity $\tilde{c}[S, T]$ (capacity $\hat{c}[S, T]$) is the minimum (maximum) among all $s - t$ cuts as a minimum (maximum) cut.

The maximum (minimum) flow problem in a network $G = (N, A, l, c, s, t)$ can be solved in two phases:

($P1$) establishing a feasible flow f, if it exists;

($P2$) from a given feasible flow f, establishing the maximum flow $\tilde{f}$ (minimum flow $\hat{f}$)

Theorem 2.1. *Let $G = (N, A, l, c, s, t)$ be a network, $[S, T]$ a $s - t$ cut and f a feasible flow with value v. Then*

$$v = f[S, T] = f(S, T) - f(T, S) \tag{2.4a}$$

and therefore, in particular,

$$\hat{c}[S, T] \leq v \leq \tilde{c}[S, T] \tag{2.4b}$$

Theorem 2.2. *Let $G = (N, A, l, c, s, t)$ be a network, $[\tilde{S}, \tilde{T}]$ a minimum $s - t$ cut and $[\hat{S}, \hat{T}]$ a maximum $s - t$ cut. Denote the values of a maximum flow $\tilde{f}$ and a minimum flow $\hat{f}$ by $\tilde{v}$ and $\hat{v}$, respectively. Then*

$$\tilde{v} = \tilde{c}[\tilde{S}, \tilde{T}] \tag{2.5a}$$

and

$$\hat{v} = \hat{c}[\hat{S}, \hat{T}] \tag{2.5b}$$

By convention, if $(x, y) \in A$ and $(y, x) \notin A$ then we add arc (y, x) to the set of arcs A and we set $l(y, x) = 0$ and $c(y, x) = 0$. For the maximum (minimum) flow problem, the residual capacity $\tilde{r}(x, y)(\hat{r}(x, y))$ of any arc $(x, y) \in A$, with respect to a given flow f, is given by $\tilde{r}(x, y) = c(x, y) - f(x, y) + f(y, x) - l(y, x)$ ($\hat{r}(x, y) = c(y, x) - f(y, x) + f(x, y) - l(x, y)$). The network $\tilde{G}(f) = (N, \hat{A})$ ($\hat{G}(f) = (N, \hat{A})$) consisting only of the arcs with $\tilde{r}(x, y) > 0$ ($\hat{r}(x, y) > 0$) is referred to as the residual network with respect to flow f for maximum (minimum) flow problem.

There are two approaches for solving maximum flow problem [1]:

($\tilde{1}$) using augmenting directed path algorithms from source node s to sink node t in residual network $\tilde{G}(f)$;

($\tilde{2}$) using preflow-push algorithms starting from source node s in residual network $\tilde{G}(f)$.

There are three approaches for solving minimum flow problem [4]:

($\hat{1}$) using decreasing directed path algorithms from source node s to sink node t in residual network $\hat{G}(f)$;

($\hat{2}$) using preflow-pull algorithms starting from sink node t in residual network $\hat{G}(f)$;

($\hat{3}$) using augmenting directed path algorithms from sink node t to source node s or using preflow-push algorithms starting from sink node t in residual network $\widetilde{G}(f)$.

In this paper we present the augmenting directed path algorithms from sink node t to source nodes s with dynamic tree implementations for solving minimum flow problems.

All the algorithms from Table 2.1 are augmenting path algorithms, i.e. algorithms which determine augmenting directed path from source node s to sink node t (by different rules) in residual network $\tilde{G}(f)$ and then augment flows along these paths.

We have $n = |N|$, $m = |A|$, $\bar{c} = max\{c(x, y)|(x, y) \in A\}$.

Augmenting path algorithms	Running time
General augmenting path algorithm	$O(nm\bar{c})$
Ford-Fulkerson labelling algorithm	$O(nm\bar{c})$
Gabow bit scaling algorithm	$O(nmlog(\bar{c}))$
Ahuja-Orlin maximum scaling algorithm	$O(nmlog(\bar{c}))$
Edmonds-Karp shortest path algorithm	$O(nm^2)$
Ahuja-Orlin shortest path algorithm	$O(n^2m)$
Dinic layered networks algorithm	$O(n^2m)$
Ahuja-Orlin layered networks algorithm	

Table 2.1. Running times for augmenting path algorithms

Actually, any augmenting path algorithm terminates with optimal residual capacities. From these residual capacities we can determine a maximum flow by following expression:

$$\tilde{f}(x, y) = l(x, y) + max\{0, c(x, y) - \tilde{r}(x, y) - l(x, y)\}$$

3. Augmenting path algorithms for minimum flows. Dynamic tree implementations

A dynamic tree is an important data structure that researchers have used extensively to improve the worst-case complexity of several network algorithms. In this section we describe the use of this data structure for augmenting direct path algorithms from sink node t to source node s.

The dynamic tree data structure maintains a collection T of node-disjoint rooted trees, each arc with an associated value. Each rooted tree is a directed in-tree with a unique root. Each node x (except the root node) has a unique predecessor, which is the next node on the unique path in the tree from that node to the root. We store the predecessor of node x using a predecessor index $p(x)$. If $y = p(x)$, we say that node y is the predecessor of node x and node x is a successor of node y. These predecessor indices uniquely define a rooted tree and also allow us to trace out the unique direct path from any node back to the root. The descendants of a node x consist of the node itself, its successors, successors of its successors and so on. We say that a node is an ancestor of all of its descendants. Notice that, according to our definitions, each node is its own ancestor and descendant.

This data structure supports the six operations obtained by performing the following six procedures:

$ROOT(x)$: finds the root of the tree containing node x;
$VALUE(x)$: finds the value of the tree arc leaving node x;
if node x is a root node, then it returns the value ∞;
$ANCES(x)$: finds the ancestor u of x with the minimum value $VALUE(u)$;
in case of a tie, chooses the node u closest to the tree root;
$CHANGE(x, \overline{w})$: adds a real number $\overline{w}$ to the value of every arc along the directed path from node x to $ROOT(x)$;
$LINK(x, y, w)$: combines the tree containing tree root x and tree containing node y the predecessor of node x and giving arc (x, y) the value w;
$DELET(x)$: break the tree containing node x into two trees by deleting the arc joining node x to its predecessor; we perform this operation when x is not a tree root;

The following important result lies at the heart of the efficiency of the dynamic tree data structure[1].

Theorem 3.1. *If q is the maximum number of nodes in any tree, a sequence of k tree operations, starting with an initial collection of singleton trees, requires a total of $O(klog(k+q))$ time.*

The dynamic tree implementation stores the values of tree only implicitly. Storing the values implicitly allows us to update the values in only $O(logq)$ time.

Before describing the augmenting directed path algorithms for sink node t to source node s with dynamic tree implementation for minimum flow problem, we introduce some definitions. In the residual network $\widetilde{G}(f)$, the distance function with respect to a given flow f is a function $\tilde{d}' : N \to N$ from the set of nodes N to the nonnegative integers N. We say that a distance function $\tilde{d}'$ is valid if it satisfies the following conditions:

$$\tilde{d}'(s) = 0 \qquad (3.1a)$$

$$\tilde{d}' \leq \tilde{d}'(y) + 1 \text{ for every arc } (x, y) \in \tilde{A} \qquad (3.1b)$$

We refer to $\tilde{d}'$ as the distance label of node x. We say that distance labels are exact if for each node x, $\tilde{d}'(x)$ equals the length of the shortest directed path from node x to source node s in the residual network $\tilde{G}(f)$. If $\tilde{d}'(x) = \tilde{d}'(y)+1$ we refer to arc (x, y) as an admissible arc, otherwise inadmissible. We refer to a directed path from sink node t to source node s consisting entirely of admissible arcs as an admissible directed path.

The following results are very important:

Theorem 3.2. *An admissible directed path in the residual network $\tilde{G}(f)$ is a shortest directed path from the sink node t to the source node s.*

Theorem 3.3. *If $\tilde{d}'(t) \geq n$, the residual network $\tilde{G}(f)$ contains no directed path from the sink node t to the source node s.*

These theorems can be proved in a manner similar to the proof of distance function $\tilde{d}$ for maximum flow problem [1] or similar to the proof of distance function $\hat{d}$ for minimum flow problem [4].

We can determine exact distance labels $\tilde{d}'(x)$ for all nodes in $O(m)$ time by performing a backward breadth first search of the residual network $\tilde{G}(f)$ from node source s to node sink t.

How we might use the dynamic tree data structure to improve the computational performance of augmenting directed path algorithms for sink node t to source node s? Let us use the variant of Ahuja-Orlin shortest path algorithm as an illustration. The following basic idea underlies the algorithmic speed-up. In the dynamic tree implementation, each arc in the rooted tree is an admissible arc. The value of an arc is its residual capacity.

Figures 3.1 and 3.2 give a formal statement of the algorithm.

The first two procedures, TADV and TRET, are straight forward, but the TAUG procedure requires some explanation. If node u is an ancestor of sink node t with the minimum value of $VALUE(u)$ and among such nodes in the directed path, it is closest to the source node s, then value $VALUE(u)$ gives the residual capacity of the augmenting path.

The procedure $CHANGE(t, -w)$ implicitly updates the residual capacities of all the arcs in the augmenting directed path. This augmentation might cause the capacity of move than one arc in the directed path to become zero. The WHILE loop identifies all such arcs, one by one and deletes them from the collection of rooted trees T.

Theorem 3.4. *The TAP algorithm correctly computes a minimum flow.*

```
ALGORITHM TAP;
BEGIN
  let f be a feasible flow in network G;
  determine the residual network $\tilde{G}(f)$;
  compute the exact distance labels $\tilde{d}'(x)$ in $\tilde{G}(f)$;
  let $\tilde{T}$ be the collection of all singleton nodes;
  x:=t;
  WHILE $\tilde{d}'(t) < n$ DO
    BEGIN
      IF exists admissibile arc (x,y) in $\tilde{G}(f)$
        THEN TADV(x)
        ELSE TRET(x);
      IF x=s
        THEN TAUG;
    END;
END.
```

FIGURE 3.1. Dynamic tree implementation for the variant of Ahuja-Orlin shortest path algorithm.

Proof. The TAP algorithm is same as the variant of Ahuja-Orlin shortest path algorithm except that it performs the procedures TADV, TRET and TAUG differently using trees. Notice that, knowing a feasible flow, we determine a minimum flow from the source node s to the sink node t by establishing a maximum flow from the sink node t to the source node s [4]. □

Theorem 3.5. *The TAP algorithm solves the minimum flow problem in $O(nmlogn)$ time.*

Proof. Using simple augments, we can show that the algorithm performs each of the six tree operations $O(nm)$ times. It performs each tree operations on a tree of maximum size n. The use of Theorem 3.1 establishes the result. □

4. EXAMPLE

Consider the network $G = (N, A, l, f, c, s, t)$ given in figure 4.1 with $s = 1$ and $t = 4$, which shows the lower bounds, feasible flows and capacities next to the arcs. The residual network $\tilde{G}(f)$ is shown in figure 4.2, which represents the distance labels next to the nodes and residual capacities next to the arcs.

Figure 4.3 shows the collection of singleton nodes $\tilde{T}$. The algorithm starts with the singleton tree containing only the sink node 4. Suppose that algorithm selects admissible arc $(4, 2)$ and it performs the procedure $TADV(x)$ for $x = 4$, $x = 2$. The algorithm obtains a rooted tree that contains both the sink and source nodes (see figure 4.4).

```
PROCEDURE TADV(x);
BEGIN
  LINK(x,y,$\tilde{r}(x,y)$);
  x:=ROOT(y);
END;
```

```
PROCEDURE TRET(x);
BEGIN
  $\tilde{d'}(x) := min\{\tilde{d'}(y) + 1|(x,y) \in \tilde{A}\}$;
  FOR $(z,x)$ $\in$ $\tilde{T}$ DO
    DELET(z);
  x:=ROOT(t);
END;
```

```
PROCEDURE TAUG;
BEGIN
  u:=ANCES(t);
  w:=VALUE(u);
  CHANGE(t,-w);
  WHILE VALUE(u)=0 DO
    BEGIN
      DELET(u);
      u:=ANCES(t);
    END;
  x:=ROOT(t);
  update the residual network $\tilde{G}(f)$;
END;
```

FIGURE 3.2. Procedures of TAP algorithm

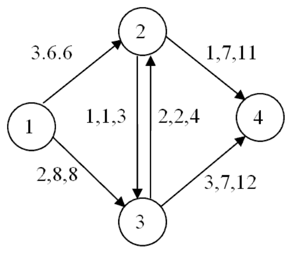


FIGURE 4.1

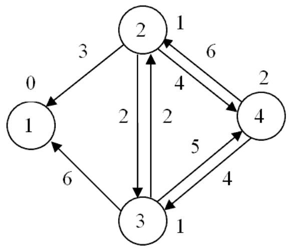


FIGURE 4.2

The procedure $TAUG$ determines the collection $\tilde{T}$ of node-disjoint rooted trees from figure 4.5, $x = 2$ and residual network $\tilde{G}(f)$ from figure 4.6. But node 2 has no outgoing admissible arc; so the algorithm performs the procedure $TRET(2)$, which increases the distance label of node 2 to 2 ($\tilde{d'} = (0,2,1,2)$), determines the rooted trees $\tilde{T}$ from figure 4.3 and $x = 4$.

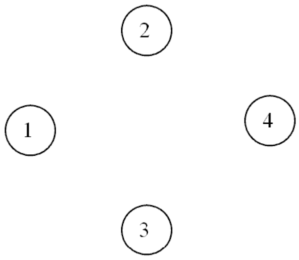


Figure 4.3

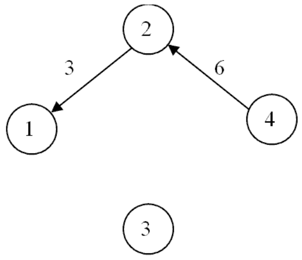


Figure 4.4

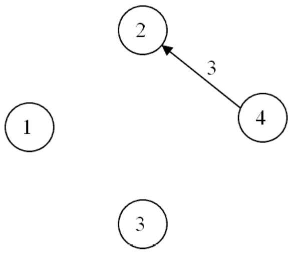


Figure 4.5

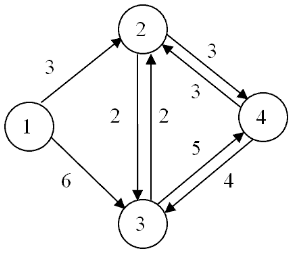


Figure 4.6

Figure 4.7 shows the last residual network $\tilde{G}(f)$ and figure 4.8 shows the lower bounds, minimum flows and capacities next to the arcs.

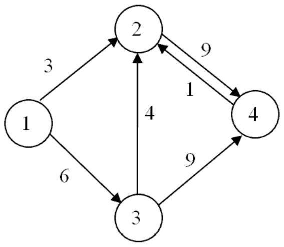


Figure 4.7

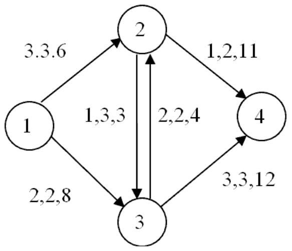


Figure 4.8

References

[1] R. Ahuja, T. Magnati and J. Orlin, *Network Flows. Theory, algorithms and applications*, Prentice Hall Inc., Englewood Cliffs, NJ, 1993.

[2] R. Ahuja, J. Orlin and R. Tarjan, *Improved time bounds for the maximum flow problem*, SIAM Journal on Computing, 18(5) (1989), pp. 939-954.

[3] J. Bang-Jensen, G. Gutin, *Digraph: Theory, Algorithms and Applications*, Springer-Verlag London Limited, 2001.

[4] E. Ciurea, L. Ciupala, *Sequential and parallel algorithms for minimum flow*, Journal of Applied Mathematics and Computing, 15 1-2 (2004), pp. 53-75.

[5] E. Ciurea, O. Georgescu, *Minimum flows in unit capacity networks*, Analele Universitatii Bucuresti, XLV (2006), pp. 11-20.

[6] A. Goldberg, R. Tarjan, *A new approach to the maximum flow problem*, Journal of ACM, 35 (1988), pp. 921-940.

[7] D. Sleator, R. Tarjan, *A data structure for dynamic trees*, Journal of Computer and System Sciences, 24 (1983), pp. 362-391.

Department of Computer Science, Faculty of Mathematics and Informatics, Transilvania University of Braşov, Str. Iuliu Maniu nr. 50, 500091, Braşov RO

E-mail address: {e.ciurea, o.georgescu}@unitbv.ro, mihaiiolu@yahoo.com

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LIII**, Number 1, 2008

SIGNEDINTERSECTION - A NEW ALGORITHM FOR FINDING THE INTERSECTION OF TWO SIMPLE POLYGONS

ANDREEA SABAU

Abstract. The operation of determining the intersection of two polygons is one of the most important operations of computational geometry. This paper presents a new algorithm called `SignedIntersection` which founds the intersection of two simple polygons, convex or concave, without holes, in two steps. The first step is using a sweep line in order to find the intersection points and segments that form the polygons' intersection. This data is enriched with additional values that indicate the way the sweep line traverse the segments, if the vertexes are given in counter-clockwise order. Such an additional value is a sign, positive or negative. The final step consists in building the result using the data determined by the previous step. The result of the intersection of two (possible) non-convex polygons may be empty or may consist of one or more polygons, convex or concave.

1. Introduction

Determining the intersection of two polygons (in the 2D space) is one of the basic operations of the computational geometry. This operation is usually used within spatial data systems (like GIS), CAD and computer graphics.

The algorithms that determine the intersection of two polygons usually receive as input data two convex polygons [4, 6], or two polygons such as at least one of them is not convex (it is concave) [2, 3]. More, there are also algorithms that can handle polygons which may have holes [8]. The algorithms from the first category are easier to implement, based on simpler computations. The one presented in [6] is based on computing the convex hull of the two input polygons. The convex hull of the two polygons contains two segments that do not belong to any of the two polygons; starting with one of them, the vertexes (and the edges) of the polygons are traversed toward the other convex hull's considered segment in order to determine the intersection

Received by the editors: May 1, 2008.

2000 *Mathematics Subject Classification.* 68U05.

1998 *CR Categories and Descriptors.* I.3.5. [**Computing Methodologies**]: Computer Graphics – *Computational Geometry and Object Modeling.*

(which is also a convex polygon). Some algorithms belonging to the second category are splitting the input polygons into convex parts and apply a method for convex polygons [1, 7] (usually - trapezoids), followed by the reunion of the intermediate results. The paper [8] describes an algorithm that runs in three steps in order to find the intersection of two polygons, possibly with holes. First, the intersection points of the two input polygons are found using a sweep line. Some navigational data (numerical data) is associated with each intersection point during the second step. The last part of the algorithm determines the intersection by traversing the polygon edges and intersection points.

This paper presents another algorithm that determines the intersection of two simple (non-self-intersecting) polygons, convex or concave. The next section presents the data structures used by the algorithm and the algorithm itself.

2. The `SignedIntersection` Algorithm

An algorithm called `SignedIntersection` that builds the intersection of two simple, convex or concave, polygons is presented in this section. This algorithm is original, according to the author's knowledge, in the way it processes and analyzes data, the used data structures, and the manner in which the result is determined. This algorithm was implemented in order to be used within the 3SST relational data model [5]. Therefore this algorithm works with data stored within relations on a Microsoft SQL-Server [5] and it is written in the Transact-SQL language.

The algorithm's input data is given by two simple polygons, convex or concave. According to the 3SST relational data model, a polygon P is represented by the list of its vertexes, given in the counter-clockwise order.

Let R and Q be the two polygons considered as input data, where R = (R_1, R_2, ..., R_n) and Q = (Q_1, Q_2, ..., Q_m). R_i, i:=1..n, n≥3, are the R's vertexes, and Q_j, j:=1..m, m≥3, are the Q polygon's m vertexes. The two coordinates of a vertex V will be noted as V.x, and V.y respectively.

The main step of the `SignedIntersection` algorithm consists in sweeping the plane with a line (the sweep line) parallel with the Oy axis, beginning with the R's or Q's vertex with the minimum x coordinate. If there are (at least) two such points the whole ensemble can be rotated so that only one vertex has the minimum x coordinate. As the sweep line is moving toward the vertex with the maximum x coordinate, the segments that are forming the intersection's final result are determined. The last step of the presented algorithm consists in analyzing the segments previously found and building the intersection polygons (none, one or more such polygons).

2.1. The `SignedIntersection` Algorithm's Data Structures.

The data structures used during the execution of the `SignedIntersection` algorithm are arrays of which elements contain the following elements:

`Points [PID, x, y, PgID]`

- `PID` - one polygon's vertex identifier (`PID` is the unique identifier in `Points` table of the 3SST relational data model [5]),
- `x, y` - the coordinates of the point identified by `PID`,
- `PgID` - the identifier of the polygon of which one of the vertexes is `PID`.

`Segments [SgID, PID1, PID2]`

- `SgID` - segment identifier,
- `PID1, PID2` - the end-points of the segment given by `SgID`. `PID1` is the start end-point and `PID2` is the final end-point, in accordance with the counter-clockwise order of the polygons' vertexes.

`Intersections [PID, x, y, SgID1, SgID2]`

- `PID` - the identifier of an intersection point between two segments of polygons R, and Q respectively,
- `x, y` - the coordinates of the point identified by `PID`,
- `SgID1, SgID2` - the identifiers of the segments from whose intersection resulted `PID`.

`Overlapping [PID1, PID2]`

- `PID1, PID2` - the identifiers of the end-points of the overlapping between two segments or a vertex and a segment of the two polygons; if the overlapping is given by a single point, then `PID1` = `PID2`; `PID1` and `PID2` always represent polygon vertexes.

`OrderPoints [PID]`

- `PID` - the identifier of a point from `Points` or `Intersections` data structures.

`Stack [position, SgID, sgn, sw_y]`

- `SgID` - a segment identifier, from R or Q,
- `sgn` - the way the segment is swept according to the sweep line; if the sweep line goes first through the start end-point of the segment, then `sgn` = +1, else `sgn` = -1,
- `sw_y` - the y coordinate of the intersection between the segment and the sweep line (the x coordinate of the intersection is given by the position of the sweep line).

`Results [SgID, sgn, PID1, PID2, checked]`

- `SgID` - a segment identifier, from R or Q,
- `sgn` - the way the segment is swept according to the sweep line (this value is taken from one `Stack`'s entry),
- `PID1, PID2` - the end-points of the segment or of the part of the segment which is included in the final result of the polygons' intersection,
- `checked` - a Boolean value, used in order to build up the final result; indicates if the segment has already been analyzed or not.

In this point the following observation has to be made. The `PID` values from the `Points` and `Intersections` lists are unique (such a value uniquely identifies an intersection point or one of the polygons' vertexes).

The `OrderPoints` and `Stack` lists are sorted. Without describing the operations, any insertion or deletion in / from these lists is maintaining the sorting order. The items of the `OrderPoints` structure are sorted according to the x coordinate of the point identified by `PID`. The list `OrderPoints` contains all the polygons' vertexes and intersection points after the sweep line completely swept both polygons. If there are two (or more) points having the same value of their x coordinate, the sorting is made according to the y coordinate. Specifically, the order in which the points of the `OrderPoints` list are stored denotes the order in which the sweep line encounters the two polygons' vertexes. The segments of the `Stack` list are sorted in the ascending order of the `sw_y` values. The `Stack` list contains at a specific moment all the two polygons' segments which are currently intersected by the sweep line. These segments' order is given by the y coordinate of their intersection points with the sweep line.

2.2. **The SignedIntersection Algorithm.** The algorithms that determine intersection points or intersection surfaces usually use the sweep line technique. The novelty and the name of the presented algorithm come from the importance of the way the polygon's edges are traversed by the sweep line. Also, there are identified four types of vertexes of a polygon and the segments in the `Stack` list are managed according to these types. The types of vertexes and the manner in which the segments are inserted, updated, or deleted in / from `Stack` are outlined next.

Let R be a simple concave polygon given by the list of vertexes R = (R_1, R_2, R_3, R_4, R_5, R_6, R_7) (see figure 1) and a sweep line parallel with the Oy axis. The order in which the vertexes are traversed by the sweep line is (R_1, R_7, R_2, R_6, R_4, R_3, R_5). The manner in which the edges of this polygon are managed in `Stack` is exemplified next. R_1 is called **extreme left vertex** of the R polygon, in which case both segments that leave from R_1 are inserted in `Stack`: R_1R_2 is inserted with `sgn`=+1 because the way the sweep line traverses it is from the initial end-point (R_1) through the final end-point (R_2); R_1R_7 is inserted with `sgn=-1` because the seep line traverses the segment as the vertexes would be taken in clockwise order (the segment R_1R_2 is inserted in `Stack` under the segment R_1R_7 because, even if R_1 belongs to both segments, R_2 has a smaller y coordinate than R_7). Next, R_7 is called **transition point with negative sign**, in which case the segment finished by R_7 is replaced with R_7R_6 in `Stack` (also with `sgn=-1`). R_2 is called **transition point with positive sign**, in which case the segment finished by R_2 is replaced with the segment that starts at R_2 (the segment R_1R_2 is replaced with the segment R_2R_3 with `sgn=+1`). R_7R_6 is replaced with R_6R_5 in `Stack` when the vertex R_6 is encountered. R_4 is an other extreme left vertex of R, therefore the segments R_4R_3 and R_4R_5 are inserted in `Stack` between R_2R_3 and R_6R_5. The vertex R_3 is called **extreme right vertex**. This vertex indicates the moment when the sweep line has just finished traversing the segments R_2R_3 and R_4R_3, therefore they are deleted from `Stack`. R_4R_5 and R_6R_5 are deleted from `Stack` when R_5 is swept finally by the sweep line. Having two polygons, their individual segments are managed within the list `Stack` in the same manner as presented above.

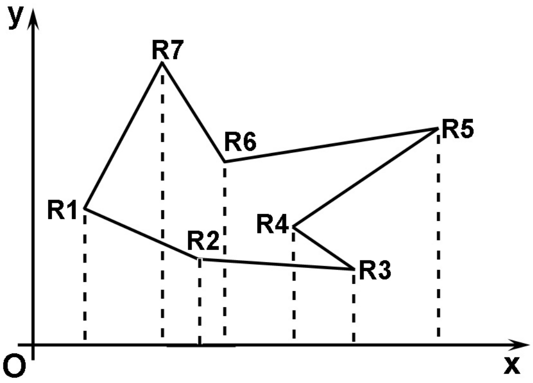


FIGURE 1. The simple concave polygon R given by R = (R_1, R_2, R_3, R_4, R_5, R_6, R_7), where the vertex R_1 has the minimum x coordinate.

The operation of determining the intersection's result is described in the algorithm presented below. The algorithm also handles the case when two segments of the two considered polygons are overlapping.

The following functions are considered to exist, without specifying their implementation details:

- no_elem(L) - returns the number of elements of one of the list structures (L) used by the SignedIntersection algorithm,
- y_int(SgID, SWL) - returns the y coordinate of the intersection point between the segment identified by SgID and the sweep line SWL,
- coord_x(PID) - determines the x coordinate of the point identified by PID (from the list Points or Intersections),
- sgn(SgID) - returns the sign of the segment SgID during the sweep line's movement through the polygons' vertexes.

Three routines are presented next: InitializeStructures initializes the lists Points, Segments, Intersections, Stack, and Results; SignedIntersection determines the intersection of the two given polygons; ShowResults analyzes and prints the intersection's result.

```
 InitializeStructures(R, Q)
// Input:
//    R, Q - polygons given by their vertexes lists,
//    R = (R1, R2, ..., Rn) and Q = (Q1, Q2, ..., Qm),
//    and identified by IdR, and IdQ respectively

   // Initialize the lists Points, Segments, Intersections,
   // Stack, Results
   For each point Ri, i:=1..n, do
     Insert the entry (Ri.PID, Ri.x, Ri.y, IdR) in Points
     Insert the entry (Ri.PID) in OrderPoints
     Let SgID be a new segment identifier
     Insert the entry (SgId, Ri.PID, R(i+1) MOD n.PID) in
       Segments
   endfor
   // The points Qi, i:=1..n, are handled in the same manner
   // as the vertexes of R
end InitializeStructures

 SignedIntersection(R, Q)
// Input:
//    R, Q - polygons given by their vertexes lists,
//    R = (R1, R2, ..., Rn) and Q = (Q1, Q2, ..., Qm),
//    and identified by IdR, and IdQ respectively

   InitializeStructures(R, Q)

   crt_pos:=1
```

```
// the position of the current point within OrderPoints

While crt_pos < no_elem(OrderPoints) do
  SWL:=coord_x(OrderPoints[crt_pos].PID)
  // SWL is the sweep line and its current position is
  // given by the x coordinate of the current point
  // (polygon vertex or intersection point)
  For each i:=1..no_elem(Stack) do
    // Update the y coordinate of the intersection point
    // between SWL and each segment within Stack
    Stack[i].sw_y:=y_int(Stack[i].SgID, SWL)
  endfor

  If OrderPoints[crt_pos].PID is in Points then
    // This point is a polygon vertex
    Let s1 be the identifier of the segment from Segments
      for which OrderPoints[crt_pos].PID is the initial
      end-point
    Let s2 be the identifier of the segment from Segments
      for which OrderPoints[crt_pos].PID is the final
      end-point
    Let Pg_complem be the identifier of the polygon such as
      OrderPoints[crt_pos].PID is not its vertex

    If (∄ k such as Stack[k].SgID=s1) and
      (∄ l such as Stack[l].SgID=s2) then
      // If neither the segment s1 nor s2 are in Stack
      // then OrderPoints[crt_pos].PID is extreme left
      // vertex of the polygon
      HandleExtremeLeftVertex(s1, s2, Pg_complem)
    else
    If (∃ k such as Stack[k].SgID=s1) and
      (∃ l such as Stack[l].SgID=s2) then
      // If both segments s1, s2 are in Stack then
      // OrderPoints[crt_pos].PID is extreme right vertex of
      // the polygon
      Delete from Stack the entries where SgID in s1, s2
      // The segments that have been "terminated" by the
      // current point are deleted from Stack
    else
    If (∄ k such as Stack[k].SgID=s1) and
      (∃ l such as Stack[l].SgID=s2) and
      (∄ m such as
      Overlapings[k].PID1=OrderPoints[crt_pos].PID) then
      // The current point is transition point with positive
      // sign and it is not an overlapping point
      HandleTransitionPointPositiveSign(s1, s2)
    else
    If (∃ k such as Stack[k].SgID=s1) and
      (∄ l such as Stack[l].SgID=s2) and
      (∄ m such as
      Overlapings[m].PID1=OrderPoints[crt_pos].PID) then
      // The current point is transition point with negative
      // sign and it is not an overlapping point
      HandleTransitionPointNegativeSign(s1, s2)
    else
      // The current point is a transition point and an
      // overlapping point
      If ∃ k such as
       Overlapping[k].PID1=OrderPoints[crt_pos].PID then
       Let s1 and s2 be the segments of the two polygons
        for which the current point is a right end-point
        (maximum x coordinate) or an overlapping point
       HandleInitialOverlappingPoint(s1, s2)
      else
      If ∃ k such as
       Overlapping[k].PID2=OrderPoints[crt_pos].PID
       and Overlapping[k].PID1<>Overlapping[k].PID2 then
       Let s1 and s2 the segments of the two polygons for
        which the current point is a left end-point
        (minimum x coordinate) or an overlapping point.
        Consider s1.PID2.x<s2.PID2.x
       HandleFinalOverlappingPoint(s1, s2)
      endif; endif
    endif; endif; endif; endif
    CheckIntersections(Stack)
  else // The current point is an intersection point
       // and it is not an overlapping point
    Let s1 and s2 be the segments that determined the
      current intersection point
    Invert the segments s1 and s2 in Stack
    // The sweep line reached the intersection point of the
```

```
      // two segments
      HandleIntersectionInResults(s1)
      HandleIntersectionInResults(s2)
      CheckIntersections(Stack)
    endif
    crt_pos:=crt_pos+1
  endwhile

  If no_elem(Results)>0 then
    ShowResults(Results)
  endif
End SignedIntersection

 HandleExtremeLeftVertex(s1, s2, Pg_complem)
// Input:
//    s1, s2 - two segments that have as left end-point the
//    current point
//    Pg_complem - the polygon that does not contain s1 and s2

  Insert the entry (s1, +1, y_int(s1, SWL)) in Stack
  Insert the entry (s2, -1, y_int(s2, SWL)) in Stack
  If the point OrderPoints[crt_pos].PID is inside the
    Pg_complem polygon then
    // (inside the polygon) or (on the frontier and the
    // other end-points of s1 and s2 are inside the
    // polygon)
    Let PID1, PID2 be the initial and final end-points of s1
    Insert the entry (s1, +1, PID1, PID2) in Results
    Let PID1, PID2 be the initial and final end-points of s2
    Insert the entry (s2, -1, PID2, PID1) in Results
  endif
end HandleExtremeLeftVertex

 HandleTransitionPointPositiveSign(s1, s2)
// Input:
//    s1, s2 - the segments for which the current point makes
//    a transition with positive sign; the current point
//    finishes the sweeping of s2 and starts the sweeping of
//    s1

  // Replace s2 with s1 in Stack
```

```
   Stack[k].SgID:=s1, where k such as Stack[k].SgID:=s2
   If ∃ k such as Results[k].SgID=s2 and Results[k].PID2 is
     the initial end-point of the s1 segment then
     Let PID1, PID2 be the initial and final end-points of s1
     // Insert s1 in Results
     Insert the entry (s1, Stack[k].sgn, PID1, PID2) in
       Results
   endif
end HandleTransitionPointPositiveSign

 HandleTransitionPointNegativeSign(s1, s2)
// Input:
//    s1, s2 - the segments for which the current point makes
//    a transition with negative sign; the current point
//    finishes the sweeping of s1 and starts the sweeping of
//    s2

   // Replace s1 with s2 in Stack
   Stack[k].SgID:=s2, where k such as Stack[k].SgID:=s1
   If ∃ k such as Results[k].SgID=s1 and Results[k].PID2
     is the final end-point of the s1 segment then
     Let PID1, PID2 be the initial and final end-points of s2
     // Insert s2 in Results
     Insert the entry (s2, Stack[k].sgn, PID2, PID1) in
       Results
   endif
end HandleTransitionPointNegativeSign

 HandleInitialOverlappingPoint(s1, s2)
// Input:
//    s1, s2 - two segments that are overlapping; the current
//    point represents the initial end-point of the
//    overlapping segment

   // Handle s1 and s2 as in the case when the point current
   // is an intersection point
   If ∃ k such as Results[k].SgID=s1 and
     Results[k].PID2 is a polygon vertex then
     Results[k].PID2:=OrderPoints[crt_pos].PID
   endif
   If ∃ k such as Results[k].SgID=s2 and
```

```
     Results[k].PID2 is a polygon vertex then
     Results[k].PID2:=OrderPoints[crt_pos].PID
   endif
   If Overlapping[k].PID1<>Overlapping[k].PID2 then
     Delete from Stack the entries where SgID in s1, s2
     Insert in Stack the segment determined by
       Overlapping[k].PID1 and Overlapping[k].PID2
     Insert in Results two entries corresponding to the two
       segments that overlap, given by Overlapping[k].PID1
       and Overlapping[k].PID2
   endif
end HandleInitialOverlappingPoint

 HandleFinalOverlappingPoint(s1, s2)
// Input:
//    s1, s2 - two segments that are overlapping; the current
//    point represents the final end-point of the overlapping
//    segment

   Insert in Stack two entries corresponding to the
     segments s1 and s2
   If sgn(s1)=+1 then
     If sgn(s2)=+1 then
       Insert in Results entries corresponding to s1 and s2
     else
       Insert in Results entry corresponding to s2
     endif
   else
     If sgn(s2)=+1 then
       // Insert no entry in Results
     else
       Insert in Results entry corresponding to s1
     endif
   endif
end HandleFinalOverlappingPoint

 HandleIntersectionInResults(s)
// Input:
//    s - one of the two segments involved in an intersection;
//    the current point is the corresponding intersection
//    point
```

```
  // Handle segment s in Results
  If ∃ k such as Results[k].SgID=s and
    Results[k].PID2 is a polygon vertex then
    // The part of the segment s that belongs to the
    // intersection is finished by the current point
    Results[k].PID2:=OrderPoints[crt_pos].PID
  else
    If (∄ k such as Results[k].SgID=s) or
      (∃ k such as Results[k].SgID=s1 and Results[k].PID2 is
      an intersection point) then
      Insert the entry
        (s, Stack[l].sgn, OrderPoints[crt_pos].PID, P) in
        Results, where l such as Stack[l].SgID=s, and P is
        the final end-point of s (if Stack[l].sgn=+1) or P
        is the initial end-point of s1 (if Stack[l].sgn=-1)
    endif
  endif
end HandleIntersectionInResults
```

The CheckIntersections routine checks whether two neighbor segments in Stack intersect when they belong to different polygons. If a new intersection point Pi is found, such as Pi is not an end-point, then it is inserted into list Intersections. If Pi is a polygon vertex such as it is the final end-point of a segment s1 and the initial end-point of a segment s2, and belongs to a segment's interior of the other polygon (s) then Pi is inserted in Overlapping as PID1. If s2 and s are overlapping then the pair of the two points that determine the overlapping segment is inserted in Overlapping.

```
 ShowResults(Results)
// Input:
//    Results - the list of segments that determine the result
//    of the two polygons' intersection

  While ∃ k such as Results[k].checked=false do
    // Initiate the building of a new intersection polygon
    Let P1 be the point PID1 or PID2 from Results which has
      the minimum x coordinate and is an end-point of a
      segment s' with sgn=+1.  Let l be the position of P1
      in Results
    Let P_init:=P1
    Let P2 be the other end-point of s′
    While P_init P2 do
      Print P1, P2
      Results[k].checked:=true
      P1:=P2
      Find l′ such as Results[l′].checked=false and
        Results[l′].PID1=P1 or Results[l′].PID2=P1
    endwhile
    Show P1, P2
  endwhile
end ShowResults
```

The SignedIntersection algorithm determines the intersection of two simple polygons, where the intersection can be empty or can consists of one or more simple polygons.

3. Conclusions and Future Work

In this paper a new approach to determine the intersection between two simple polygons has been proposed. The presented algorithm uses the well-known technique of the sweep line and assigns a special sign value to each of the polygons' edges in order to find the intersection result. Also, the sign of each segment that belongs to the intersection's frontier is used to build the polygons' intersection. The SignedIntersection algorithm and the corresponding data structures are easy to implement on top of the 3SST relational data model.

It is intended to extend the SignedIntersection algorithm in order to be able to determine the intersection of two polygons with or without holes.

References

[1] M. de Berg, M. van Kreveld, M. Overmars, O. Schwarzkopf, *Computational Geometry, Algorithms and Applications*, Springer-Verlag, Berlin, 1997.

[2] A. Margalit, G. D. Knott, *An Algorithm for Computing the Union, Intersection or Difference of Two Polygons*, Computers & Graphics, Vol. 13 (2) (1989), pp. 167-184.

[3] J. O'Rourke, *Computational Geometry in C*, Cambridge University Press, 1993.

[4] J. O'Rourke, C.B. Chien, T. Olson, D. Naddor, *A New Linear Algorithm for Intersecting Convex Polygons*, Computer Graphics and Image Processing, No. 19 (1982), pp. 384-391.

[5] A. Sabau, *The 3SST Relational Model*, Studia Universitatis "Babes-Bolyai", Informatica, Vol. LII (1) (2007), pp. 77-88.

[6] G. T. Toussaint, *A Simple Linear Algorithm for Intersecting Convex Polygons*, The Visual Computer, Vol. 1 (1985), pp. 118-123.

[7] B. Zalik, G. Clapworthy, *A Universal Trapezoidation Algorithm for Planar Polygons*, Computers & Graphics, Vol. 23(3) (1999), pp. 353-363.

[8] B. Zalik, M. Gombosi, D. Podgorelec, *A Quick Intersection Algorithm for Arbitrary Polygons*, SCCG98 Conf. on Comput. Graphics and it's Applicat. (1998), pp. 195-204.

Faculty of Mathematics and Computer Science

Babes-Bolyai University, Cluj-Napoca
E-mail address: `deiush@cs.ubbcluj.ro`

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LIII**, Number 1, 2008

INTRODUCING A NEW FORM OF PARAMETRIC POLYMORPHISM IN OBJECT ORIENTED PROGRAMMING LANGUAGES

IANCU MIHAI CĂPUTĂ AND SIMONA MOTOGNA

Abstract. Nowadays software development tools have to provide effective means of data manipulation with minimal development time. As types represent the meaning of raw data, this paper focuses upon taking types to another level in an object oriented dynamically type-safe programming language, in order to increase language flexibility and productivity.

1. Introduction

Mainstream object oriented programming languages, such as .NET languages or Java, support both kind of type-checks (static and dynamic).

Static type checking or static typing means that the typechecker, which may or may not be part of the compiler, performs an analysis over the source-code at compile time to ensure that certain type-constraints are not being violated.

In C#, for instance, static type checks are made when resolving the overloading of methods, or when performing an upcast (casting from derived type into base type). Dynamic type checking (also known as runtime type checking) characterizes a dynamically typed programming language which is one where type constraints are being checked at runtime. In C#, dynamic type checks are made when a downcast (casting from base type into a derivate type) is performed, in order to ensure that the backing object, that is cast into the derivate type, is an instance of the derivate type or of another type that derives from that derivate type. This check can only be done at runtime, except some scenarios in which a smart compiler can figure out that the cast is possible.

By taking the dynamically-typing even further, productivity can be increased considerably. This can be achieved by introducing the concept of

Received by the editors: May 5, 2008.

2000 *Mathematics Subject Classification.* 68N15, 68N19.

1998 *CR Categories and Descriptors.* D.3.3 [**Language Constructs and Features**]: Data types and structures, Polymorphism.

Type-unbound variables which induces a new form of parametric polymorphism. We propose a programming language prototype, called X Language or shortly X, that will incorporate this new form of parametric polymorphism with clear benefits in productivity. It targets the .NET framework 2.0 and the syntax is almost identical to C# 2.0s. This language is under development. We will often make references to C#, which is very popular amongst developers, but these references are traceable in most of the OOPL on the mainstream. (Java, Delphi.NET, C++.NET etc.)

This paper is organized as follows: The second paragraph is introducing the notion of type unbound variables, and then in paragraph three we discuss its relation with parametric polymorphism, describing in several examples the role of type unbound variables in implementing paramatric polymorphism. Section 4 discusses different aspects that should be taken into consideration when introducing this feature in a programming language. Paragraph 5 refers to some details regarding the implementation of type unbound variables, targeting a virtual machine, and in the end some ideas for our future work on this langauge.

2. Introducing type unbound variables

In C#, when you declare a variable, you must specify its type. This is a hint for the compiler so that it will be able to enforce type-safety. That variable will be bound to its type throughout its entire scope. This means that you wont be able to change its type at runtie. Please note that eventhough a variable of type B can have a backing object of type D, which is a derivate of B, conceptually speaking that variable is of type B (from a compilers point of view). Type unbound variables refer to variables that arent bound to a certain type. At the moment i of execution the variable has the type T_i and at the moment $i+1$ it can have any other type, T_{i+1} which is not necessarily ad-hoc polymorphic with T_i.

In almost every object oriented programming language, there are system classes used to manipulate the concept of Type, such that information about types (system or user defined ones) is accessible at runtime throughout instances of this classes. In C# there are mechanisms (such as reflection [4]) for creating objects based on information held by these instances.

This is the common way to create instances of a variable type at runtime. Sometimes this turns to be quite tedious and the usage of `System.Type` class in order to do that does not intervene to the programmer in a natural manner. This class hints us more to a class schema or an object runtime inspector rather than to a usable type.

In the **X** language, which implements the concept of type unbound variables, the behavior of `System.Type` class instances is similar to the one of a type identifier. We can use an instance of the `System.Type` class (which holds meta-informations about a type) just as if it was a type, as shown in the following example:

Example 1: Usage of a type unbound variable (a)

```
System.Type T;
T = int;
T a = 10; //a is a valid integer with the value of 10
```

3. Type unbound variables and parametric polymorphism

Polymorphism is a programming language feature that allows values of different data types to be handled using a uniform interface. Christopher Strachey identified in 1967, two fundamentally different kinds of polymorphism: ad-hoc and parametric [3].

Ad-hoc polymorphism is when the range of actual types that can be used is finite, and the combinations must be specified individually prior to use. In object-oriented programming this is implemented through class-inheritance (objects of different types can be handled uniformly through an interface or a common base class called superclass).

Parametric polymorphism enables a function or a data type to be written generically so that it can handle values identically without depending on their type[1], such that it increases the expresiviness of the language.

Firstly introduced in ML (1967), and then inherited in several other langauges, parametric polymorphisim still remains a desirable feature in a programming langauge, due to its benefits. Recently, Java and C# introduced ”generics” as a form of parametric polymorphism.

Cardelli and Wegner [1] introduced in 1985 ”bounded parametric polyorphism”, which imposes some bounds on the parameters, such as to be subtypes of a given type.

For example, in C++ parametric polymorphism is implemented with templates or in C# with generics, as presented in Example 2.

Example 2: C# Generics

```
List<int> myList = new List<int>();
myList.Add(1);
myList.Add(2);
myList.Add(3);
```

This kind of parametric polymorphism is resolved statically at compile time: ”In .NET 2.0, generics have native support in IL (intermediate language) and the CLR itself. When you compile generic C# server-side code, the compiler compiles it into IL, just like any other type. However, the IL only contains parameters or place holders for the actual specific types. In addition, the metadata of the generic server contains generic information. The client-side compiler uses that generic metadata to support type safety. When the client provides a specific type instead of a generic type parameter, the client's compiler substitutes the generic type parameter in the server metadata with the specified type argument. This provides the client's compiler with type-specific definition of the server, as if generics were never involved. This way the client compiler can enforce correct method parameters, type-safety checks, and even type-specific IntelliSense"[2].

In the following, we propose some use cases and present some examples that show how type unbound variables are introduced and what improvements they have on the written code.

Use case 1: In C# the abstract data type List is available as a generic type. When you use the class List to specify the type of a variable, or to derive from it, you have to specialize it by telling the compiler what kind of elements this list will handle. Parametric polymorphism comes inherently in X Language when using an instance of `System.Type` class as a type identifier for a methods return type, a methods argument or class member, as illustrated below.

Example 3:

```
class GenericList
{ protected System.Type itemType;
  public ItemType
  {
    get { return this.itemType;}
    set {this.itemType = value;}
  }
  public GenericList(System.Type itemType)
  {
    ItemType = itemType;
  }
  public void Add(ItemType item)
  {//... }
  public void Remove(ItemType item)
  { //...}
  public ItemType operator [](int index)
  {//...}
   . . .
  }
```

In the above example notice that the property `ItemType`, that represents the type of one item from the `GenericList`, is used in `Add, Remove` method declaration, and in [] operator declaration. Parametric polymorphism comes from the fact that by changing the `ItemType` of the `GenericList` instance, it will handle different lists of items, without having to change any code.

When specializing this kind of genericity (by supplying a valid `System.Type` instance for each `Type variable` in the class), you will not create a new type, but a new behavior. This approach makes the definition of generic lists more naturally.

Use case 2: Another situation in which type unbound variables can be useful is the following: Suppose that we have a class, named `MyClass`, written by a third party, leaving out of posibilities of modifying this class. We want to create a proxy for `MyClass`, named `MyClassProxy` which delegates all the methods calls to a Remote method call server (this illustrates the design pattern Proxy, and is often used in RMI - Remote Method Invocation, and RPC - Remote Procedure Call). `MyClassProxy` looks identical in terms of method signatures to `MyClass`, but those types are not ad-hoc polymorphic since instances of these types cannot be treated uniformly through an interface or a base class that exposes their methods. Suppose that we have to write down code that dynamically decides whether it uses objects of `MyClass` or objects of `MyClassProxy` and does a certain task. In order to achieve that, in C# 2.0, well either have to write the code that does the job, twice, firstly for `MyClass`, and secondly for `MyClassProxy`, or as an alternative we will have to extract that code into a generic method, but that is a bit intrusive, and sometimes it is not quite straight-forward.

In **X** programming language the solution comes from the usage of the parametric polymorphism in the form of type-unbound variables. We declare a variable of type `System.Type`, and we fill it with `MyClass` or `MyClassProxy` accordingly. Then we use that variable to declare the instances of the class `MyClass/MyClassProxy`, and operate with them just as if we dont have to decide whether to use `MyClass` or `MyClassProxy`. Example 4 shows how this approach is taken in **X**.

Example 4:

```
System.Type T;
if (bUseProxy) //we need to use the proxy class
{
  T = MyClassProxy;
}
else
{
  T = MyClass;
}
T obj; // crete an instance of MyClass or MyClassProxy
             depending on the previous decision
 // use obj no matter what type underlies it
```

Example 5: Dynamic casting - Suppose that we have a variable T of type Type. A cast will be performed at runtime from int to the value of the T variable (which is a type).

```
int a;
```

```
Type T;
T b;
//read a value for a;
if (a>0)
{
   b = float;
}
else
{
   b = double;
}
b = (T)a; //a dynamic cast is performed from int to float or
              // double, depending on the runtime value of T.
```

4. Implications of type unbound variables

When implementing this new form of parametric polymorphism several aspects must be kept in mind regarding: type safety, strong typing, threads safety and limitations of type unbound variables.

4.1. **Type safety.** For each variable that is about to be used, at the moment of execution, the underlying type must be known, otherwise a runtime exception will be thrown.

4.2. **Strong Typing.**

- Whenever a method argument is a type unbound variable, that methods overloading resolution must be done at runtime
- Member access of a type unbound variable is resolved at runtime hence all the validations upon that member must be done at runtime
- Method calls of type unbound variables are late bound
- Each operator is subject to all the constraints the methods are subject to
- When changing the underlying type of a type-unbound variable, a policy regarding the current value of the variable needs to be adopted: Since every type has a default value, the value of the variable will be reset to this default value.

4.3. **Limitations.** We have identified three restrictions that must be imposed:

- Variable types (variables of type `System.Type`) cannot be used in class hierarchies definition (cannot be used as base types)
- Variable types cannot be used to define delegate types (function pointer types), as pointed out below.

```
  class D {...}
  class A
```

```
{
   Type typeVar = D;
   class C:typeVar { ...} //this is not allowed
   delegate typeVar myDelegate(D b, A a) ;// not allowed
   delegate int myDelegate(typeVar b, A a);//not allowed
}
```

- Type instances cannot be used to define entities (variables/members) of a larger domain of visibility, as in the following sequence

```
class C
{
   private Type memberType;
   public memberType member1; //not allowed:  member1
      // has a larger domain of visibility than memberType
   protected memberType member2; //not allowed:  member2
      //has s larger domain of visibility than memberType
}
```

4.4. **Thread safety.** `System.Type` instances can be regarded as shared resources once they were used to create instances of a type. The problem of thread safety arises here.

Suppose that a type-unbound variable V is used by thread Th1, and thread Th2 wants to change the underlying type of V. The system should expose a mechanism through which the programmer could be able to ensure thread-safety.

In order to ensure this, the `System.Type` class from **X**, exposes a variable counter, which indicates the number of type unbound variables of that type. When the variable counter is 0, the type can be changed without causing any havoc. Please note that this variable counter is different than the reference counter which indicates how many instances of that type are being referenced at a moment of execution.

In a compiler implementation, the type-variables could be copied into the thread local storage (with a compiler directive the programmer is able to modify this implicit behaviour) so the programmer wont have to interogate the variable counter before changing the value of the type variable.

4.5. **Lifetime and variable storage.** The lifetime of a type-unbound variable is not determined by the lifetime of the type instance that was used to define it. Consider the sequence:

```
{
   Type T;
   T a = new T();
   StartThread(a); //start a thread and pass 'a' as parameter
}
```

When T runs out of scope, the instance that was passed to the thread doesnt get garbage collected. In the storage of variable a, a reference to the value of T will be held. When variable T runs out of scope, its reference counter gets decremented, but it will not reach down to 0 because the instance referenced by a, will be passed to a thread, hence incremented.

The type-unbound variable storage is subjected to all the policies, that the type-bound variables (the regular ones) are subjected to.

5. Implementation details

When implementing such a feature in a programming language, several aspects should be taken into consideration. Firstly, type-unbound variables are not syntactic sugar. In order to implement this feature several extensions must be supported by the core of the virtual machine and covered by the design of the compiler.

There are plenty of ways to implement such a feature, and many performance-related policies can be applied. These implementation details aim a virtual machine that will support type-unbound variables.

The compiler will not be able to fully handle the usage of variables, because their type might be unknown at compile time. This implies that the virtual machine will provide mechanisms to handle variables (such as member function call, member access, etc) as instructions built within its core. In the sequence of code below, well try to exemplify how one can implement this feature. The code is written in C++, and covers only the surface of the concept: how to implement instructions in the virtual processor, that provide method invoking and member access of type-unbound variables.

The idea behind the implementation can be understood from the definitions of the structures and the functions.

The structure *Method* holds metainfo about a method. These kind of metainfo are also usefull for reflecting upon a method. The structure *Type* holds metainfo about a type, in which the methods and the type contains can be organized as a dictionary that maps a method hash to a *Method*. The structure *Instance* contains information about an instance of a class. If the type member of this structure can be chagned, we are talking about a type unbound variable.

The function *call()* represents the call instruction supported by the virtual processor. The function *OverloadingResolution()* resolves the collision of methodes that have same *methodHash*. Two methodes have same *methodHash* if they have same name.

The function *xcall* represents the extended call instruction suported by the virtual processor. It perfoms a dynamic call.This means that the method cannot be determined at compile time, and it has to be searched through the methods of the underlying type, by the *methodHash*. This hash is determined at compile time by applying some sort of a hash function on the method name. *methodAddress* represents the method address that needs to be determined. If the type does not contain a definiton for that method, then an exception will be thrown. If the method is virtual, then we need to lookup its address into the virtual function table, still by its hash, and if the method is not virtual then its address is the one held by the *methodInfo* structure returned by the method overloading.

The *xMemberAccess* function provides access to a member returning its address from a specific variable storage. It will perform a look-up by the *memberHash*, into the *Type* of *Instance*, to find the address of the member into the *Instance* storage. This access is type checked, which means that if the *Type* does not contain a definiton for the specified member, an exception will be thrown.

```
struct Method
{
   void* address; //address of method, determined at load-time
   bool isVirtual; //whether or not the method is virtual
   //...  //any other metainfo
};
struct Type
{
   Method** methodes;
};
struct Instance
{
   Type* type; //the type of the instance
   void* vft; //virtual function table
   void* storage; //data storage
   //...//other related info
};
void call(void* method) { }
Method* OverloadingResolution(Type* type, unsigned long methodHash,
   Instance** params)
{ }
void xcall(Instance* _this, unsigned long methodHash,
   Instance** params, unsigned int paramCount)
{
   void* methodAddress = 0;
   Method* methodInfo = OverloadingResolution(_this→type,
   methodHash, params);
   if (methodInfo==0)
      throw new Exception();
   if (methodInfo→IsVirtual)
   {
        methodAddress = _this→vft[methodInfo];
    }
    else
    {
        methodAddress = methodInfo→address;
    }
    push(_this);
    for (int i=0;i<paramCount;++i)
    {
        push(params[i]);
    }
    call(methodAddress);
}
void* xMemberAccess(Instance* _this, unsigned long memberHash) { }
```

6. Future work

The fully development of the **X** Language, will illustrate this concept.

We also intend to describe the complete definition of type constraints that will allow the static type-checker to resolve some validations which are currently done by the runtime type-checker.

Also the intellisense of the development environment **X Language** will integrate in, would be developed so that it will work in the case of type-unbound variables and variable types.

7. References

[1] L. Cardelli and P. Wagner: *On Understanding Types, Data Abstraction and Polymorphism*, Technical Report CS-85-14, Brown University, Department of Computer Science, 1985

[2] Juval Lowy: *An Introduction to C# Generics*, Visual Studio 2005 Technical Articles, 2005, http://msdn2.microsoft.com/en-us/library/ms379564(vs.80).aspx

[3] C. Strachey, Fundamental concepts in programming languages. Lecture notes for International Summer School in Computer Programming, Copenhagen, August 1967

[4] T.L. Thai, Hoang Lam - *.NET Framework Essentials*, O'Reilly Programming Series, 2001

Department of Computer Science, Faculty of Mathematics and Computer Science, Babeş-Bolyai University, 1 M. Kogălniceanu, Cluj-Napoca 400084, Romania

E-mail address: ci29836@scs.ubbcluj.ro

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

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LIII**, Number 1, 2008

SOFTWARE PROCESS IMPROVEMENT AT SYSGENIC

DUMITRU RĂDOIU AND MILITON FRENŢIU

Abstract. The Capability Maturity Model (CMM) was defined by Software Engineering Institute as a mean to improve the state of Software Engineering process. Going from CMM level i to the next level i+1 is seen as a major improvement. How such an improvement was obtained at Sysgenic is presented in this paper. Also, some consequences on the teaching process are presented.

Keywords: Software Process Improvement, CMM, education

1. Introduction

The term "software crisis" was introduced by the participants at two NATO conferences held in 1968 and 1970. It was observed that for a large number of software development projects deadlines are frequently missed, cost overruns are a rule not an exception, and it is increasingly difficult to measure the project progress. It was estimated [18] that more than 50% of the development project time is spent on testing and debugging. And the final product is not error-free; on average there still can be three to five errors for every hundred statements. We all expect the results given by our programs are correct, but 71% of software products have errors during their usage [16]. It is well known that some errors are not detected by testing, and some of them are never detected. Moreover, there are projects that have never been finished [4]. It is estimated that one from three large projects was never finished [7, 16].

There is a growing requirement of new programs. But our ability to build new programs hardly keeps pace with the demand for new programs. We have all observed the permanent and rapid growth of computer usage in all fields of human activity. The need for new programs is immense and their complexity is continuously growing. There are known today programs with millions of lines of code written by hundreds of people. These more complex programs cannot be developed like the old small ones. It has become necessary

Received by the editors: May 15, 2008.

2000 *Mathematics Subject Classification.* 68N30.

1998 *CR Categories and Descriptors.* D.2.9 [**Software Engineering**]: Management – *Software process models.*

to analyze software costs over the entire life cycle of the system. It is known today that the major errors are due to errors in specifications, or poor design of the system, not to bad coding. After all, the fraction of time needed for programming is about 20% of the entire development process.

Among the factors that make software difficult four of them are inherently difficult [1, 14]. They are well known: the complexity of software, the conformity of the software product with the real world, the changeability and the invisibility of the product. The complexity of the problem affects the entire development process. It is difficult to understand the problem, to analyze it, to design and to implement the software product. And the real world cannot be changed to make the software product simpler.

The maintenance activity is a fundamental aspect of software engineering. It is a significant portion (exceeding 50%) of the total development process. But poor design and inadequacies resources threaten our ability to maintain existing programs.

To improve the state of programming activity the Software Engineering Institute approved a project to study this activity and to suggest ways of improving it. One of the project outcomes is a report [12,13] that contains the conclusions of that study and – based on the findings – suggests a number of steps considered to improve the software processes. According to this study the software companies are classified in 5 levels, defined by their performances. Key process areas specific to each level are also given.

Going from CMM level i to the next level i+1 is seen as a major improvement of the software processes of a company. In this respect CMM is seen as a guide for a continuous improvement.

Software Process Improvement (SPI) may be simply characterized by three main outcomes:

- respecting the cost and schedule of projects specified in the contracts;
- increasing the productivity;
- improving the quality of software products.

Reducing the rework is one possibility to increase software productivity, and to respect the schedule. We must build correct programs from the beginning [9, 11, 3]. As Gilb said: “Prevention is more effective than cure” [8]. Also, removing errors earlier permits to reduce the rework, and it is known today the importance of inspections (peer reviews) in this direction [6].

Building correct programs from the beginning is not a dream, today it becomes a reality. At IBM Mills introduced such a methodology, known as Cleanroom [10]. It has been successfully used for 20 years.

2. Reaching CMMi level 3 at Sysgenic

Sysgenic is a Romanian software development company with expertise in projects for financial and capital markets supplying customers in Europe and USA. The company software process improvement (SIP) started with documenting and institutionalizing ISO 9001:2000 requirements and this quality management system was certified in august 2005. The decision to implement a more professional quality management system (CMMi Level 3) was based on the need of more control over the projects based on more structured and practical project management principles. The outcomes were expected to be reflected in work performance, project visibility and control and in the end higher quality.

The Process Areas involved in CMMi Level 3 implementation are: Requirements Management (REQM), Project Planning (PP), Project Monitoring and Control (PMC), Measurements and Analysis (MA), Process and Product Quality Assurance (PPQA), Configuration Management (CM), Requirements Development (RD), Technical Solution (TS), Product Integration (PI), Verification (VER), Validation (VAL), Organizational Process Focus (OPF), Organizational Process Definitions (OPD), Organizational Training (OT), Integrated Project Management (IPM), Risk Management (RSKM), and Decision Analysis and Resolution (DAR).

First step was to set up the project team, also known as the Process Improvement Group (PIG), consisting of process oriented practitioners, with extensive experience in process design, software development and project management.

Second step was to provide them professional training (in CMMi) and documentation. PIG initial training started with an "Introduction to CMMI" SEI course, plus recent and extensive documentation on the capability maturity model integration (CMMi).

Third step consisted in initiating a "gap analysis" to document the differences (in the above mentioned areas) between what was implemented in the company and what are CMMi requirements. Based on these findings PIG initiated the design of the new internal process and started to implement them. Within the space of one year, these set of standard processes (OSSP) were institutionalized in the organization. Processes started to follow the new standards and to be documented accordingly.

After the processes were institutionalized and appeared to comply with the new requirements, the fourth step was an internal evaluation, also known as SCAMPI B. The differences between SCAMPI B findings and CMMi Level 3 required compliance were were smoothed up.

The following step is called "running in production mode": release internal process assets library, OSSP in organizational and software projects.

The on-site assessment, also called SCAMPI A, consists of

- pre-onsite period (consisting in collection and evaluation of project evidence), and
- on-site evaluation.

Following the on-site evaluation, Sysgenic achievement was recognized by Software Engineering Institute [15] in August 2007.

Now, one year later after being certified CMMi level 3, Sysgenic is following an internal QA audit on processes followed by a process improvement analysis and implementation on evaluation results. It is worth mentioning the constraints under which Sysgenic went into this SPI. Here are the most "visible" ones:

(1) making use of the already defined processes, known and largely used by employees;
(2) analyzing and deciding on the best usage of the existing tools (not always the best choice under CMMi level 3 exigency);
(3) a limited number of human resources who could be allocated to the SPI.

A first remark, following the above presented constraints, is that the newly CMMi Level 3 defined and institutionalized processes were sometimes time consuming and some even redundant. As projects are usually allocated small teams (8-10 people), any overhead generated by excessive documentation and/or training is "visible" in planning and cost and will lower the company competitiveness.

Following the first observation is that SPI never ends and the processes should be continuously reviewed and improved in successive iterations, focused mostly on:

(1) process and documentation simplification, maintaining compatibility with CMMi Level 3 and 4 requirements;
(2) identifying the most suitable tools which automate certain activities;
(3) processes institutionalization and periodic training (and re-training).

3. Conclusions

There are a significant number of lessons learned; here are some mistakes which we could have avoided.

- Make sure you have really experienced PIG team members, with a positive attitude, knowledgeable in their respective areas or expertise. During the process the PIG is usually overloaded, in a small company

they have to play multiple roles and therefore more likely to make errors. Their expertise and attitude, plus good planning as well as monitoring and control are essential in the success of the project.
- Simplify your processes to be more close to what we do, more natural to perform, to really help you in improving your organization performance.
- Develop your own simple and goal oriented metrics to document and track performance.

Software critical systems [2] require error-free programming and high quality software development processes. Obviously this requires also better educated work force, able to do this. It was a pleasant finding to learn that CMMi is taught at Petru Maior University (based in Tirgu Mures where Sysgenic HQ is located) as part of the Software Engineering course.

Attaining level 4 is, certainly, the next goal of Sysgenic. Quantitative process management and Software quality management are the Key Process Areas for this level. Continuous improvement, training and learning from our experience will help. A Software Metrics Program must be introduced to offer a quantitative feedback for improvement.

Nevertheless, we need more and more educated people as Software Engineers. And these people need a serious background from universities. Knowledge on Process Management, Verification and Validation (and consequences from the theory), Software Metrics must be present in their curricula [17].

There is a contradiction between the desire to obtain a system as quickly as possible, and to have a correct system. We need confidence in the quality of our software products. We need to educate the future software developers in the spirit of producing correct, reliable systems. For this we must teach students to develop correct programs. We are aware that usually programmers do not prove the correctness of their programs. There always must be a balance between cost and the importance of reliability of the programs. But even if the well educated people do not prove the correctness, their products are more reliable than the products of those “programmers” who never studied program correctness. Therefore, we consider that the students must listen, and pay attention to the correctness of their products.

It is unbelievable that students are superficially taught the theory of program correctness. As teachers, we must strive for a better education of the new generations of programmers. As scientists, we must look for better tools. The software development process must be based more on mathematical techniques, the formal methods must be taught and used as much as possible.

References

[1] Brooks, F.P., No Silver bullet: Essence and accidents of software engineering, IEEE Comput. 20, 4(1987), 10-19.

[2] Ricky W.Butler, Sally C.Johnson, Formal Methods for Life-Critical Software, NASA Langley Research Center, http://shmesh.larc.nasa.gov.

[3] Dromey, G., Program Derivation. The Development of Programs from Specifications, Addison Wesley, 1995.

[4] Effy Oz, When Professional Standards are LAX. The CONFIRM Failure and its lessons, Comm. A.C.M., 37(1994), 10, 29-36.

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

[6] Tom Gilb and Dorothy Graham, Software Inspection, Addison-Wesley, 1993.

[7] Gibs W.W., Software's Chronic Crisis, Scientific American, September, 1994.

[8] Gilb T., Software Inspection for the Internet Age: how to increase effect and radically reduce the cost, 2001, www.Result-Planning.com

[9] Gries, D., The Science of Programming, Springer Verlag, Berlin, 1985.

[10] Mills H., M.Dyer, and R. Linger, Cleanroom Software Engineering, IEEE Software, 4 (1987), 5, 19-25.

[11] Carol Morgan, Programming from Specifications, Springer, 1990.

[12] Paulk M.C., B.Curtis, M.B.Chrissis, C.V.Weber, The Capability Maturity Model for Software, Tech.Report, CMU/SEI-93-TR-25.

[13] Paulk M.C., B.Curtis, M.B.Chrissis, C.V.Weber, The Capability Maturity Model, Version 1.1, IEEE Software, 10(1993), 4, 18-27.

[14] Schach S.R., Software Engineering, IRWIN, Boston, 1990.

[15] http://sas.sei.cmu.edu/pars/pars_detail.aspx?a=9828 (retrieved 3^{rd} of June 2008)

[16] The Standish Group Report: Chaos, http://www.scs.carleton.ca/∼bean/PM/ Standish-Report.html

[17] ***. Computer Science Curricula at Babes-Bolyai University, www.cs.ubbcluj.ro

[18] Yourdon, E., Modern Software Analysis, Yourdon Press, Prentice Hall Buiding, New Jersey 07632, 1989

Petru Maior University, 1 Nicolae Iorga St., Târgu Mureş, Romania
E-mail address: Dumitru.Radoiu@Sysgenic.com

Babeş Bolyai University, 1 Mihail Kogălniceanu St., Cluj-Napoca, Romania
E-mail address: mfrentiu@cs.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LIII**, Number 1, 2008

MODELING OF THE IMAGE RECOGNITION AND CLASSIFICATION PROBLEM (IRC)

IOAN ISPAS

Abstract. The problem of the image recognition and classification (IRC) based on the pattern recognition is of a paramount importance in lots of domains. The present paper discusses topics related with the complexity of the algorithms for image recognition and classification. This leads to some precise statements on the computational difficulty of the problem of the image recognition and classification (IRC).

KEY WORDS: modeling, image recognition and classification, algorithm complexity

1. Defining the problem of image recognition and classification (IRC)

The automatic classification of the images is of a strategic importance in lots of domains. Its solving is based on the methods and algorithms of automatic pattern/object recognition and image classification [3].

In the following part, we will define the IRC problem:

Given an image data base (data stream) $B = \{I_1, I_2, \ldots, I_n\}$ containing a 'main character', each image incorporating only one object; given a set of descriptions of some known distinct objects $R = \{O_1, O_2, \ldots, O_k\}$; knowing that any human operator is able to recognized easily, by means of rapid visual inspection, the object in the image; the aim of the algorithm is to determine the images that contain these objects, and to classify them in $k+1$ distinct classes: $C_1, C_2, \ldots, C_k$ and C_{k+1}. The classes C_i, $i = 1, k$ will group all the images containing the objects O_i, $i = 1, k$, and the class C_{k+1} will group the images without any of the R objects.

The diagram of the IRC problem is the following:

Received by the editors: April 2, 2008.

2000 *Mathematics Subject Classification.* 93A30, 68T10.

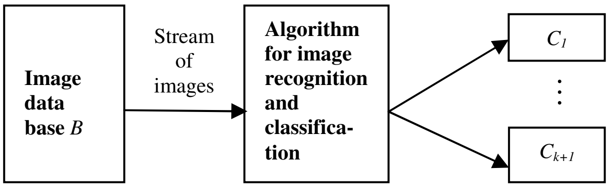


Figure 1. offers a small example consisting of eight images with ‘characters‘ [13] that can be classified into seven image classes: 1. the class of the images containing horses; 2. the class of the images containing cheetahs; 3. the class of the images containing elephants; 4. the class of the images containing airplanes; 5. the class of the images containing bears; 6. the class of the images containing eagles; 7. the class of the images ”neutral”, without recognized object. In this case, the component elements of the set of the recognized ‘objects‘ are: 1. horses; 2. cheetahs; 3. elephants; 4. airplanes; 5. bears; 6. eagles.

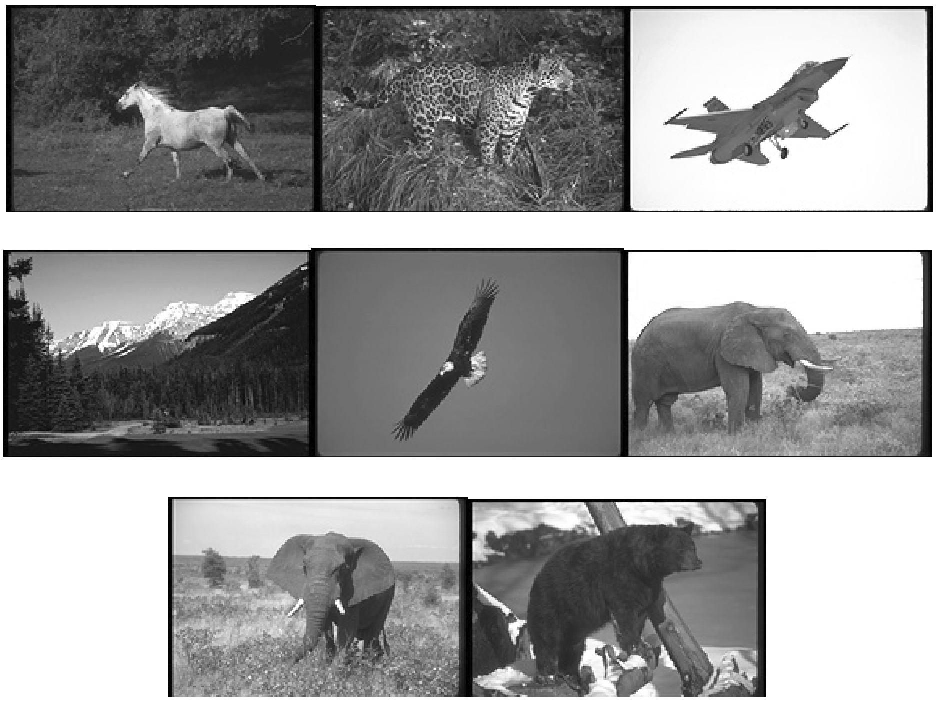


FIGURE 1. Examples of images to be classified

Having studied the specialized literature [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11] we can state that the image recognition algorithms describe a four-stepped process. Each step is essential and inevitable. The diagram of the image recognition process is the following:

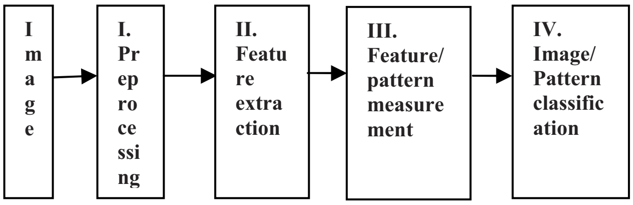


Image
I. Preprocessing
II. Feature extraction
III. Feature/ pattern measurement
IV. Image/ Pattern classification

I. Preprocessing of the image. This means the application of some DIP (Digital Image Processing) algorithms specialized in enhancing image quality [1], [3], [5], [10].

II. Feature extraction. This is the key step, the one that measures the performances and the quality of the recognition software. The discovery of the most proper features and characteristics of the recognition object is the key of the success [12], [14], [15], [16], [17], [18], [19], [20]. The $FeatureExtraction$ algorithm, the implementation of this essential step, output a feature vector description of the recognized object $(v_1, v_2, \ldots, v_n)$, not necessarily numerical.

III. Feature/pattern measurement. This step is well theoretically founded; there is a developed mathematical theory (The measure theory) which can help us select the proper and efficient n-dimensional metrics. The final result of this step is usually a one- or multi- dimensional value (a vector) percepted as the ‘distance‘ of the feature vector towards the borders of the class [1], [12], [15], [18], [19].

IV. Image/Pattern classification. This is the final step which combines the results of the prior measurements. The pattern/object - described by the feature vector - belongs to a class of images, according to certain appartenance mathematical criteria. The final result of the classification step must be the index i of the image class C_i .

2. The modeling of the problem of the image recognition and classification

The primary modeling of the problem of the image recognition and classification is an extremely difficult subject. In order to reduce its degree of difficulty, a gradual approach is indicated to be used in a step-by-step manner.

2.1. The simplified version of the problem of the image recognition and classification (sIRC).

If we consider the objects-‘characters‘ from the images as marks/signatures, that were previously inserted in the images. Consequently, every object became obviously a ”main character”, in front of the scene. Then the recognition of images leads us to the following simplified version of the IRC problem defined at the beginning:

Given an image data base (or an image stream) $B = \{I_1, I_2, \ldots, I_n\}$ containing a ‘main character‘ that marks them; given a set of descriptions of some known distinct objects $R = \{O_1, O_2, \ldots, O_k\}$; the following algorithm determines the images from B that contain these objects and classifies them in $k+1$ distinct classes: $C_1, C_2, \ldots, C_k$ and C_{k+1}. The algorithm has an image I as input and is calling the sub-algorithm *Recognition*; this algorithm decided if object O_k is contained in image I.

```
Algorithm **sIRC**(image I);
For $i = 1, k$ do
    If **Recognition** $(O_i, I)$ return $(i)$;
Return $(k + 1)$;
```

Our belief, just like its title shows, is that the simplified version **sIRC** problem is less difficult than the initial one. Unfortunately, we cannot prove rigorously this statement although the multitude of facts strongly confirms it. It is obvious that its complexity relies on the complexity of the Recognition sub-algorithm. The total complexity of the algorithm is in the worst of cases:

$$\text{WorstCase}(\textbf{Classification\_sIRC}) = k \times \mathcal{O}(\textbf{Recognition}),$$

where k is the dimension of the set of objects R. $\mathcal{O}(\textbf{Recognition})$ is the classical notation for the complexity class of the Recognition algorithm.

The **Recognition** algorithm is the clue of the **sIRC** problem. Its input is I image and the description of the recognition pattern/object O. For every object O_i it works like a validation function with the output true or false. Considering that any human operator is able to easily recognize the presence of the object O in the image I by visual inspection, based on a primary process of the mathematical modeling and formalization of the sIRC problem, we can design the following modeling:

2.2. Mathematical modeling of the sIRC problem.

Definition 1. A **searching space** is a set of data S which has to be exhaustively covered in order to find the target data x among S data.

Giving n the cardinal of the set S; considering that the exhaustive covering condition is needed, then the number of required steps (comparisons) in order to find out x is in the worst case n.

Definition 2. A **pattern** P of an (b_i-dimensional) object is the set of the contour points (laying on the external edges of the shape of the object) which delimits the space occupied by it.

The pattern P of an object is that what makes it distinguished from the environment and confers its identity.

Definition 3. An **informational content** (colorist) C of a certain object is a set of points belonging to the object, grouped together according to an association (relational) criteria.

For instance, the set of the 'interior' points of the object, the set of the points of the same color, etc. The information content (colorist) C is the visible, descriptive expression of the object.

Proposition 1. Therefore we can state that every object is uniquely defined by its pattern P and its informational content C. As the pattern P and its informational content C are described by numerical vectors, the pair (P, C) uniquely defines every object.

2.3. Introducing the auxiliary problem IRC(R).

The terms of the auxiliary problem IRC(R) are the following: this problem is particular case of the sIRC problem in which the set of objects to be recognized consists in a single type of objects R, as an image of the solid rectangle.

Any rectangle R is defined by the pair of corners $A(x_A, y_A)$ and $B(x_B, y_B)$, and by its colour C. Thus $R = R(A, B, C)$.

The problem requests to determine the subset Q of the image set containing one rectangle R.

The easiest method (considering the effort in designing the recognition algorithms) is the scanning of all the images, i.e. for every image, to check up every possible matching position of the rectangle R. The effort of recognition and classification of the images, which is directly proportional to the image resolution and indirectly proportional to the dimension of the rectangle R, will thus be huge. Redesigning this brute force approach method implies of new, more efficient methods to match the rectangle R, other than exhaustive scanning of the image. Since an image I certainly contains a rectangle R, one can question if the image I, having $M \times N \times c$ resolution, could be seen as a searching space for rectangle $R(A, B, C)$, where c is the color resolution of the image I.

Given the set of all valid coordinations $S(A, B)$ for the rectangle R in the image I. Every recognition algorithm $\mathcal{A}$ of R in I, seen from the **Turing-Church Thesis** perspective [21], can be simulated by a Universal Turing Machine U is possible. The algorithmic complexity of the Machine U is identical with the recognition algorithm $\mathcal{A}$. Moreover the Universal Turing Machine U is designed to display every pair of possible coordinates (A, B) as algorithm $\mathcal{A}$ overpasses them at its run-time, following its specific steps.

Therefore, it is obvious that the set of all valid coordinations $S(A, B)$ becomes a searching space for the algorithm $\mathcal{A}$. None of the pairs (A, B) can avoid being checked. In conclusion, the difficulty of the $IRC(R)$ problem is the same with the difficulty of finding a value x in the searching space S by exhaustively scanning. In this case, x represents the coordinate pair (A, B) while S represents the set of all valid coordinations the rectangle can have in the image.

2.4. **Reduction of the sIRC problem to IRC(R) problem.** According to Proposition 1 every recognized object O is defined by the pair of vectors (P, C), pattern and informational content. Since P and C are vectors and not singular values, the pair (P, C) denotes a rectangle in a multidimensional space. The recognition of the object O in the set of the images can be consequently regarded as the matching of the rectangle (P, C) in the corresponding searching space, resulted from the union of all the pairs of valid coordinates (P, C) of the object O in the images. Thus, the sIRC problem is reduced to the IRC(R) problem.

3. Conclusions resulting from the mathematical modeling of the sIRC problem

Conclusion 1. Generally speaking, for each Recognition(O, I) algorithm the image I becomes a searching space for the object to be recognized $O(P, C)$.

Conclusion 2. The complexity of the algorithm Recognition(O, I) is direct proportional with the dimension of the image I and with the dimension of the codes P and C :

$$\mathcal{O}(\text{Recognition}(O, I)) = \mathcal{O}(Dim(I) \times Dim(P) \times Dim(C))$$

4. Final results and conclusions

Resuming the statements before, we can reach the following final results.

Result 1. The difficulty of the simplified version of the problem sIRC is a consequence of the complexity of the algorithm Recognition(O, I).

Result 2. The essential step in the recognition process of the object O is the feature extraction of the patterns and of the information content (P, C) from the image I. Notice. This feature extraction step is inevitable because the image I is formed by a matrix of pixels, but the descriptors (P, C) are a pair of codes describing the shape and the information content of the object O.

Result 3. The complexity of the algorithm Recognition(O, I) is directly proportional with the dimension of the features (P, C) extracted by the sub-algorithm FeatureExtraction(O), the main component of the second step of the recognition process, pointed out earlier in the diagram of the image recognition process.

Result 4. The complexity of the algorithm Recognition(O, I) is given by the formula:

$$\mathcal{O}(\text{Recognition}(O, I)) = Dim(I) \times \mathcal{O}(FeatureExtraction(O))$$

where $\mathcal{O}(FeatureExtraction(O))$ is the complexity class of the $FeatureExtraction$ algorithm.

Important notice. The extraction of the color and of the pattern features from the image may imply a very consistent number of operations m (i.e. associations and relations) over the pixels within the interest zones. The complexity of the extraction algorithm of the object O becomes:

$$\mathcal{O}(\text{FeatureExtraction}\,(O))) = m \times Dim(\text{ExtractionZone})$$

Note that the determination/discrimination of the interest zones (which could contain the object) is the most important but also the most difficult step in the entire feature extraction process. This can lead to a situation wherein the recognition of an object having dimension 200×200 pixels, within an image having a resolution of 800×600 pixels and 256 colors, could require a number of operations directly proportional with the huge value 800×600×256×200×200, greater than 10^{12}.

The final conclusion about the difficulty of the image recognition and classification problem is that a proper solution of the problem and of its simplified version sIRC depends in the most direct way on the design of an efficient pattern/information content extraction sub-algorithm.

References

[1] Gonzalez R., Woods R. - Digital Image Processing, Prentice Hall, 2002, 2nd Edit.

[2] Jain A., Duin R., Mao J. - Statistical Pattern Recognition: A Review, IEEE Transactions On Pattern Analysis and Machine Intelligence, Vol. 22, No. 1, pp. 720-736, January 2000.

[3] Russ, John C., The image processing handbook, 5th ed., CRC Press, 2006.

[4] Richard O. Duda, Peter E. Hart, David G. Stork, Pattern Classification, 2nd ed., Wiley Interscience, 2000.

[5] Sankar K. Pal, Amita Pal (eds), Pattern Recognition. From Classical to Modern Approaches, World Scientific Publishing Company, 2002.

[6] Mitra Basu and Tin Kam Ho (eds), Data Complexity in Pattern Recognition, Advanced Information and Knowledge Processing, Springer-Verlag, 2006.

[7] Bahram Javidi (ed), Image Recognition and Classification. Algorithms, Systems, and Applications, CRC Press, 2002.

[8] Bernd Jahne, Digital Image Processing, 5th ed., Springer, 2002.

[9] Yali Amit, 2D Object Detection and Recognition. Models, Algorithms and Networks, The MIT Press, 2002.

[10] Sing-Tze Bow, Patern Recognition and Image Preprocessing, 2nd ed., Marcel Dekker Ltd., 2002.

[11] Ioan Ispas, The image recognition and classification, a four-steped modeling, Proc. 2nd International Conf. on European Integration - Between Tradition and Modernity, Petru Maior University, Tîrgu Mureş, Sept 20-21, pp. 724-730, 2007.

[12] Kian-Lee Tan, Beng Chin Ooi, Chia Yeow Yee - An Evaluation of Color-Spatial Retrieval Techniques for Large Image Databases, Multimedia Tools and Applications, 14, 55-78, 2001, Kluwer Academic Publishers.

[13] elib.cs.berkeley.edu/photos/classify/

[14] Oge Marques, Borko Furht - Muse: A Content-Based Image Search and Retrieval System Using Relevance Feedback, Multimedia Tools and Aplications, 17, 21-50, 2002, Kluwer Academic Publishers.

[15] Y. Alp Aslandogan, Clement T. Yu, Ravishankar Mysore, Bo Liu - Robust content-based image indexing using contextual clues and automatic pseudofeedback, Multimedia Systems 9: 548-560 Springer-Verlag 2004.

[16] Kian-Lee Tan, Beng Chin Ooi, Chia Yeow Yee - An Evaluation of Color-Spatial Retrieval Techniques for Large Image Databases, Multimedia Tools and Applications, 14, 55-78, 2001, Kluwer Academic Publishers.

[17] A. Srivastava, A.B. Lee, E.P. Simoncelli, S.-C. Zhu - On Advances in Statistical Modeling of Natural Images, Journal of Mathematical Imaging and Vision 18: 17-33, 2003 Kluwer Academic Publishers.

[18] Jörg Dahmen, Daniel Keysers, Hermann Ney and Mark Oliver Güld - Statistical Image Object Recognition using Mixture Densities, Journal of Mathematical Imaging and Vision 14: 285-296, 2001, Kluwer Academic Publishers.

[19] Martin Heczko, Alexander Hinneburg, Daniel Keim, Markuswawryniuk - Multiresolution similarity search in image databases, Digital Object Identifier (DOI) 10.1007/s00530-004-0135-6, Multimedia Systems 10: 28-40, Springer-Verlag 2004.

[20] Wei-Ying Ma, B. S. Manjunath - NeTra: A toolbox for navigating large image databases, Multimedia Systems 7: 184-198 (1999) Multimedia Systems, Springer-Verlag, 1999.

[21] Stanford Encyclopedia of Philosophy, The Church-Turing Thesis, first published Jan 8, 1997; substantive revision Aug 19, 2002. http://plato.stanford.edu/entries/church-turing/

N. Iorga 1, Tîrgu-Mureş
E-mail address: `ispas@science.upm.ro`

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LIII**, Number 1, 2008

GREENLIFE – A MMORPG THAT STIMULATES AN ECOLOGICAL BEHAVIOR

MIRCEA CIMPOI, RADU MEZA, DIANA ZOICAŞ, CRISTINA CIUHUŢĂ, AND DAN SUCIU

Abstract. In recent years, MMOGs (*Massively Multiplayer Online Games*) had become popular. In the same time, more and more ways of implementing interactions between real world and game's virtual worlds are developed. These facts gave us the idea to develop GreenLife application, as a solution of involving people in recycling and ecological activities. GreenLife is a Massively Multiplayer Online Role Playing Game and its basic idea is recycling business management. The game resources are increased based on players behavior in their real life.

Key words: multiplayer game, online gaming, role-playing game.

1. Introduction

Massively Multiplayer Online Role-Playing Games (MMORPGs) are a new class of *Multi-User Domains* (MUDs) - online environments where multiple users can interact with each other and achieve structured goals. The first MUD - an adventure game in a persistent world that allowed multiple users to log on at the same time - was created in 1979 by Roy Trubshaw and Richard Bartle [1].

Even if MUDs descended from *Role-Playing Games* (RPGs), the two genres emerged around the same time and co-evolved beginning in the early 70's and became popular during the 80's. Both games allowed users to create characters based on numerical attributes (like power, dexterity, intelligence) and combat-oriented roles (ie. warrior, archer, healer, cleric, druid etc) with different strengths and weaknesses. Game-play typically revolved around a combination of interactive story-telling and logistical optimizations under the guise of slaying monsters and attaining higher levels and skills.

Received by the editors: June 1, 2008.

2000 *Mathematics Subject Classification.* 68N01, 68U20.

1998 *CR Categories and Descriptors.* I.2.1 [**Computing Methodologies**]: Artificial Intelligence – *Applications and Expert Systems - Games*; I.6.8 [**Computing Methodologies**]: Simulation and Modeling – *Types of Simulation - Gaming*.

In RPGs, a designated Game Master controlled the outcome of events based on dice-rolls and references to charts and tables. In MUDs, this was controlled by the server. As the graphical and processing capabilities of the modern personal computer increased, and as accessibility to the Internet became widely available, it became possible in the early 90's to build MUDs with graphical front-ends.

Ultima Online, launched in 1997, is recognized to be the first MMORPG - a persistent, graphical, online environment that allowed thousands of users to be logged on at the same time. The second MMORPG, *EverQuest*, launched in 1999 was the most popular MMORPG in North America for more than 5 years. [8] Today hundreds of MMORPGs are available over the internet, some of them with millions of registered users and thousands online users in every moment. According to statistics presented on Gamespy.com, for *Half-Life* and *Counterstrike* games there are more than 50.000 users online everyday and Lineage games had more than 2.000.000 (paying) subscribers at the end of 2004. The average age of MMORPG players is round to 26, with a range from 11 to 68. The lower and upper quartile boundaries are 19 and 32 respectively. [7]

In the same time, the connection between the virtual worlds created in MMORPGs and the real life became stronger and stronger:

- users with very good abilities in playing a particular game are selling virtual items (weapons, buildings or technologies) or even their own accounts via e-Bay;
- game creators are selling different real objects that allow game users (based on some codes) to unlock game facilities;
- certain amounts of virtual resources could be obtained in exchange for real money.

The popularity of MMORPGS and the various possibilities of influencing the virtual world through real world actions gave us a perfect context to imagine a way to involve people in recycling activities. We have imagined a virtual world modeled around the recycling concept and business management. The solution that we proposed is called GreenLife. Its core consists of a MMORPG and two distinct components that provide two ways of supporting the game from the outside world:

- *ShopSmart* program: using your eco-card to get some money for eco-friendly shopping;
- *Recycling* - get in-game resources for equivalent amounts of recyclable materials taken to Collecting Centers.

2. GreenLife description

The game site opens with a game setting (planet Earth depleting its resources, extracting more and more each day, recycling very little) and description (what are the objectives of a player and how they can develop their recycling business).

The basic idea of the game is recycling business management. Each player starts with an amount of money. The first thing a player has to do is to buy one or several locations. Locations are defined as sectors of the areas corresponding to cities. Each city has a certain amount of waste that can be recycled. Each sector represents a percentage of the city's recyclable waste. Depending on the size of the city and implicitly on the amount of waste, each location (sector) has a price.

After buying one or several locations, the player buys/builds recycling facilities at each location. These facilities can be upgraded as the game develops. A facility converts waste into a certain amount of recycled resources per day (depending on the technology level of the facility). These resources (glass, tin, wood and plastic) can be sold for money at the market (the market will fluctuate daily depending on the amount of that kind of resource sold overall the preceding day). The market will use real-world recycled resource prices as reference. As the game develops, the player can buy new locations and build new facilities in neighboring areas thus expanding the profits. Also, for each facility, the user can build one specialization annex that boosts the output of one specific resource (glass, tin, wood and plastic).

Players can interact with each other by chatting, forming alliances and competing for better locations (takeover). A player can try to takeover a certain location from another player. The prerequisites for this action are: the owner of the location the player wants to takeover is within two levels (overall technology level) of him and the hostile player either already has a location in that city or is part of an alliance within which one player already has a location in that city.

The player can protect himself against takeovers by investing in community support (can be achieved by giving out some of the money for recycling education at the community level, also increases the amount of recyclable waste for that location) and profitability. The outcome of a takeover action is computed depending on the technology level and profitability of the hostile player versus the community support and profitability. When a takeover succeeds, the hostile player obtains the respective location but not the facility. This can be bought (and downgraded 1 or 2 levels) from the previous owner for a certain amount of money (which the previous owner receives). The hostile player also has the option of building a new facility from scrap and not paying for the existing one.

3. GreenLife architecture

Most of commercial MMORPGs have been implemented as client server systems where the global game state consisting of positions and states of all players is managed in a centralized way. The centralized control has advantages in keeping security high and implementation easy. The model used for our MMORPG is based on a desktop client application. The decision stays on the advantages gained by using Presentation Foundation and on the need to have an executable proof of concept version. Figure 1 contains the architectural diagram of GreeLife system. The main components of this diagram are:

- *GreenLife MMORPG* - available as a desktop application version (in future it will be replaced with a web application version);
- *GreenLife Game Service* - WCF web service that facilitates the access to the database.
- *GreenLife Mobile (Collecting Center, Shopping Center)* - the mobile device version of the game. Communication will be achieved using Mobile Service

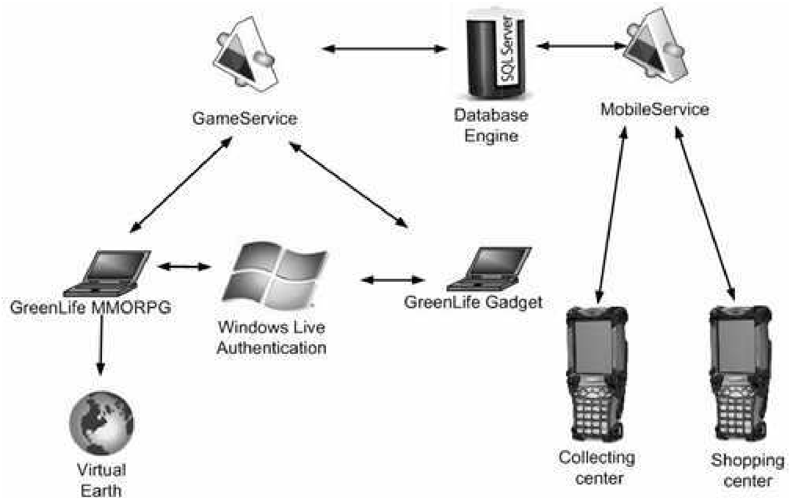


Figure 1. GreenLife architecture

Green Life database has the following structure:

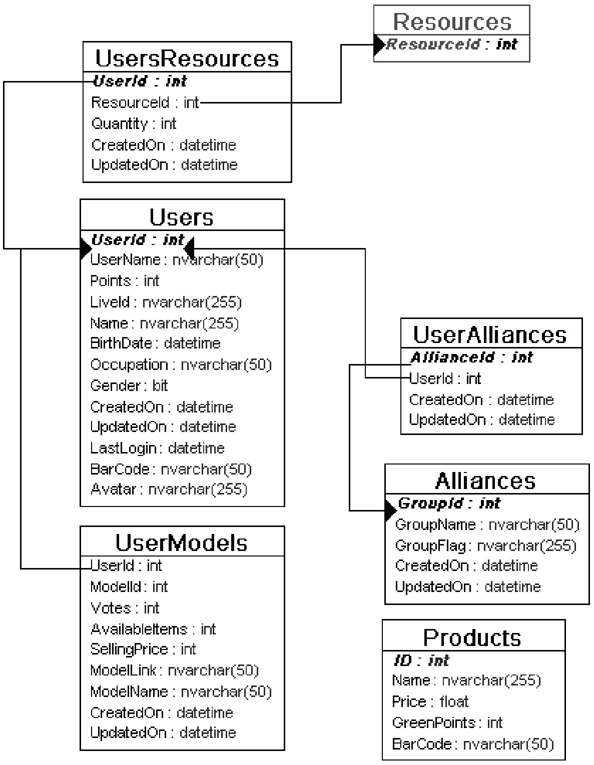


Figure 2. GreenLife user management tables

Users, *Alliances* and *UserAlliances* are tables that contain information about GreenLife registered users and groups of interests formed by them during game play.

UserModels stores 3D models of factories build / owned by a specific user.

UserResources table contains amounts of specific resource (glass, tin, wood and plastic) owned by a user. The records of *UserResources* together with Users.Points field help users to develop and increase their recycling business.

Products is an isolated table which contains the list of items that can increase game points of an user if they are obtained from certain point of sales (like hypermarkets, supermarkets or petrol station chains) involved in GreenLife system.

Locations, *SectorOfLocations*, *Resources* and *ResourceInLocations* describe the waste quantities and their position on Earth map. Each user will own one or more factory models (stored in *FactoryModels* table) that will exploit the waste of a specific location sector.

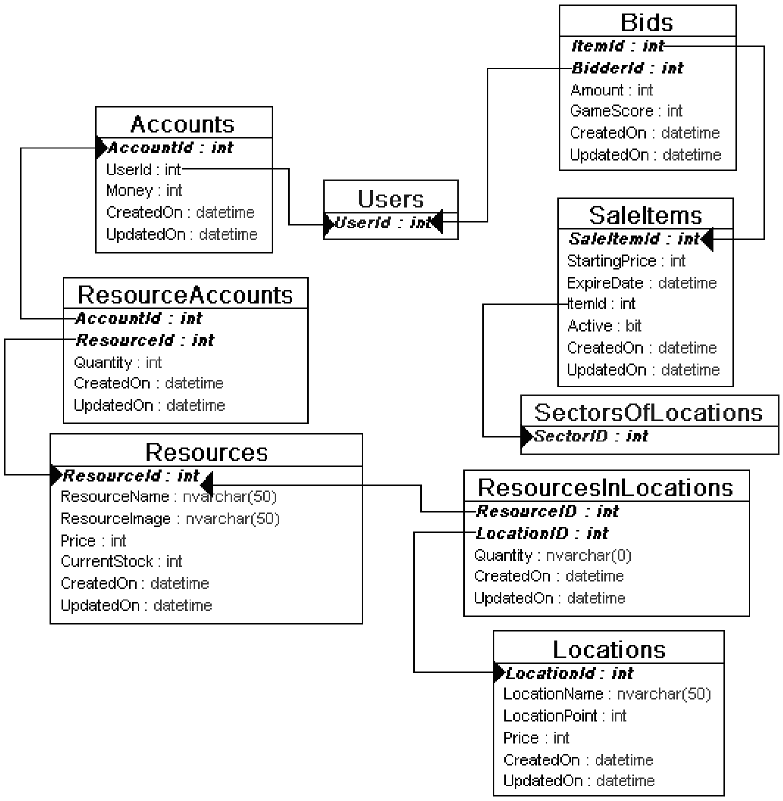


Figure 3. GreenLife resources/points management tables

Accounts and *ResourceAcounts* define the type and quantity of resources and money owned by a user and which is not visible for other competitor in the game.

Factories, *FactoryModels*, *FactoryModelAnnexes* and *TechnologyLevels* store 3D models descriptions used for graphical representation of factories and their annexes and their capacity and cost of exploiting the waste from a certain location sector.

Different technologies have been used in our implementation. We mention here *Windows Presentation Foundation* - that provides the ease of creating a rich user experience, with minimum of code behind, *Windows Communication Foundation* - for reliability and security of data transfers, *Microsoft Live Labs Volta* and *Virtual Earth v6 API* - for game map representation and *3dvia ShapeIt* for building tridimensional models of recycling factories.

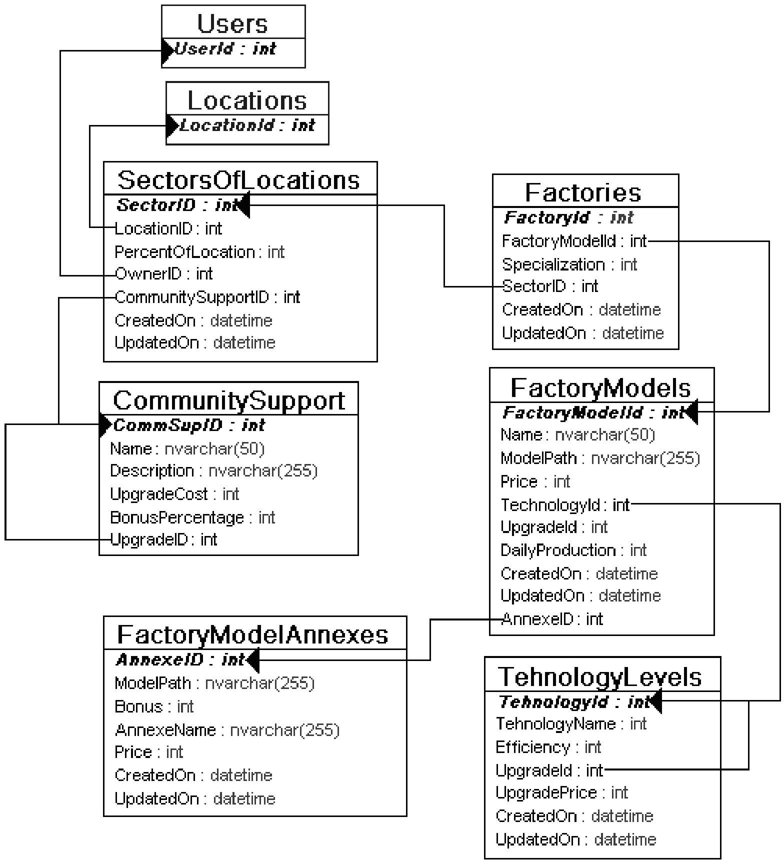


Figure 4. GreenLife map locations management tables

4. Conclusions

GreenLife MMORPG solution is characterized by the following novelties:

- it is a new type of MMORPG, considering the ecological and economical perspective;
- it redirects game dependence towards ecological education;
- it is a stronger bound between real life and virtual life through interaction with economic agents and the involvement of non-playing relatives and acquaintances;
- a new model of eco-game of great impact with the young and many others. The existing solutions of this kind were limited to simple action and a small target, preschools through teens in general.

We think that GreenLife MMORPG solution is interesting as a commercial project, mainly because of the interaction with the real world, and the advantages it presents in the collaboration with hypermarket networks and recyclable waste collecting centers. Offering low implementation costs, and the chance to promote the game easily (advertising in collaboration with markets), our solution seems a good opportunity for a successful and fast growing business. We believe that the catchy story behind the game and the attractive graphics, together with the rewards from the real world will contribute to the success of the idea.

References

[1] Bartle, R., "Early MUD History", 1990.
Available at http://www.mud.co.uk/richard/mudhist.htm

[2] Chris Chambers, Wu-Chang Feng, Sambit Sahu, Debanjan Saha, "Measurement-based Characterization of a Collection of On-line Games", Proceedings of the Internet Measurement Conference on Internet Measurement Conference, pp. 1-1, 2005

[3] T. Iimura, H. Hazeyama, Y. Kadobayashi, "Zoned Federation of Game Servers: a Peer-to-peer Approach to Scalable Multiplayer Online Games", Proceedings of the 3rd Workshop on Network and System Support for Games (NETGAMES 2004), 2004

[4] Yugo Kaneday, Hitomi Takahashiy, Masato Saitoy, Hiroto Aiday, Hideyuki Tokuda, "A Challenge for Reusing Multiplayer Online Games without Modifying Binaries", Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games, pp. 1-9, 2005

[5] R. Koster, "Online World Timeline". 2002
Available at http://www.legendmud.org/raph/gaming/mudtimeline.html

[6] Shinya Yamamoto, Yoshihiro Murata, Keiichi Yasumoto, Minoru Ito, "A Distributed Event Delivery Method with Load Balancing for MMORPG", Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games, 2005

[7] Nicholas Yee, "The Demographics, Motivations, and Derived Experiences of Users of Massively Multi-User Online", Graphical Environments, 2004

[8] Nicholas Yee, "The Psychology of Massively Multi-User Online Role-Playing Games: Motivations, Emotional Investment, Relationships and Problematic Usage", appeared in Avatars at Work and Play: Collaboration and Interaction in Shared Virtual Environments, edited by R. Schroder and A. Axelsson, London: Springer-Verlag, 2005

Department of Computer Science, Babeş-Bolyai University, 1 M. Kogălniceanu St., RO-400084 Cluj-Napoca, Romania

E-mail address: cm20294@scs.ubbcluj.ro

E-mail address: mr20338@scs.ubbcluj.ro

E-mail address: zd29939@scs.ubbcluj.ro

E-mail address: cc20297@scs.ubbcluj.ro

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

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LIII**, Number 1, 2008

ROBOSLANG – CONCEPT OF AN EXPERIMENTAL LANGUAGE

OVIDIU SERBAN

Abstract. There are very few platforms trying to unify the programming process of robots and none of them is concerned with building an extensible language that would allow anyone to have access to the hardware. RoboSlang's main goal is to allow programmers to build easily new modules without needing any hardware programming knowledge and their concern should be to create better algorithms for some well-known problems.

Keywords: Robots, Programming on embedded device, Programming languages, Communication

1. Introduction

Programming on different types of platforms (such as different hardware devices and operating systems) was always a difficult task and when trying to program non standard devices or mobile systems without a proper language or platform can be a very a very dangerous approach. Middleware platforms [12] were developed to unify the developing process, but they are only the starting point for the programming process and not all the problems were being solved. There are some problems that you may need to solve, even if you program on top of a Middleware platform, such as unifying the data feeds(communications and sensors data) or planning tasks in a distributed way.

This paper presents the concept of an experimental language and the way it could be implemented using some of the programming languages (such as Java, C, C++ or SQL). It is trying to answer some usual questions about robots communication and solve some of the basic problems of exchanging data. There are some projects oriented on the intelligent side of the robots, trying to search for a perfect algorithm that solves a problem, but there are many communication issues and hardware problems that should be solved in a different manner. RoboSlang is an entry point for every other project that

Received by the editors: October 9, 2007.

2000 *Mathematics Subject Classification.* 68T05, 91E45.

1998 *CR Categories and Descriptors.* I.2.6 [**Learning**]: – *Concept learning.*

intends to do something intelligent with some hardware and it should not be focused on the platform programming issues.

Also there should be a transparent way of exchanging information and the intelligent devices should have a platform ready for programming and for further extension. RoboSlang is trying to offer a solution for this issue. Many industrial project desire to achieve communication between the production machines, but they are, in most of the cases not able to do so. What if there existed a platform able to do these tasks such that the main concern of the industrial programmers would be a better algorithm to improve the production and not a hardware issue for some component ?

This platform is designed to be scalable by the ability to add as many dialects as needed. So if you need a domain specific dialect you will not need to implement all of the transmission mechanism from its base point, you should only extend RoboSlang for what you need.

There are many hardware incompatibilities and so many different types of architectures that only a transparent layer for every kind of communication should solve the problem, so RoboSlang proposes a unified way to transmit messages between devices.

The paper is organized as follows: The motivation and the real problems that should be handled when trying to program such platform are described in Section 1. Section 2 contains the actual description of the language. The two main dialects of RoboSlang are presented in Sections 3 and 4. Section 5 contains a few of the implementation problems and some programming suggestions. Section 6 concludes the paper.

2. Motivation of the implementation of such a platform

2.1. **What would robots say to each other?** The robots can exchange information like sensors data or positioning diagrams. This information has a huge potential in a network considering the fact that many robots and devices do not have powerful hardware or sensors to work with. An entire theory about distributed computing [13] was developed, but there is no standard protocol of communication between two devices that are able to exchange information.

Not only robots and smart devices need to communicate results, also the humans would like to communicate with the robots in a “human-like“ language and if they are capable of doing that, why would not be any other smart living creature capable of communicating with a robot or a smart device.

Imagine a situation when a cat would like to ask your robot to clean all the mess inside its sleeping place. Why would not that be possible? If you consider that a cat is capable of thinking and expressing its thoughts then there should not be a problem to exchange the information.

2.2. **Hardware and other communication problems.** Hardware compatibilities are a real issue. There are so many possibilities for exchanging information that even trying to keep a collection with all the data for them is a problem. The platform will try to offer support to any further implementations so that the hardware level will be transparent.

For example if you consider different types of wireless protocols [2], you will see that some of them are incompatible even if they are supported by the same device and you do not have to mention problems related to incompatible devices that support the same protocols. Also related to wireless problems there are some issues for data transmission in a hostile environment. We should consider that our device should be able to transmit data even if the environment is unfriendly.

Suppose that all the hardware problems are not so difficult to solve, there are also some software and communication problems. For instance two platforms support different type of data, one is able to deal with big quantity of information and other is very slow and cannot process such information. What can we do ? Fortunately the network research area gives us some answers, but there are some problems that still remain unsolved.

Another problem is related to listening process, when you search for another device to exchange information. If you do this from time to time, it is most probable to miss some of the partners. If you do this permanently then you could waste valuable energy that could be used to perform other tasks. In the real life there are some situations when you establish a synchronization moment with another partner, but in robotic like situations this is not a good way to discover other devices. You should be able to discover and exchange information as much as you can and as fast as you can, not to wait for a certain moment.

Therefore the hardware and communication problem still remains open, but in this area a lot of research is done so RoboSlang would be able to apply certain fixes as fast are discovered by other researchers. The main concern on the developing process should be the platform and not to solve certain hardware or software problems.

3. RoboSlang

This section contains the description of the proposed language (called RoboSlang).The language itself should be as light as possible. It must contain a discovery procedure, that should also involve a handshake mechanism and a procedure for choosing a dialect. The entire process can be described in a step by step manner :

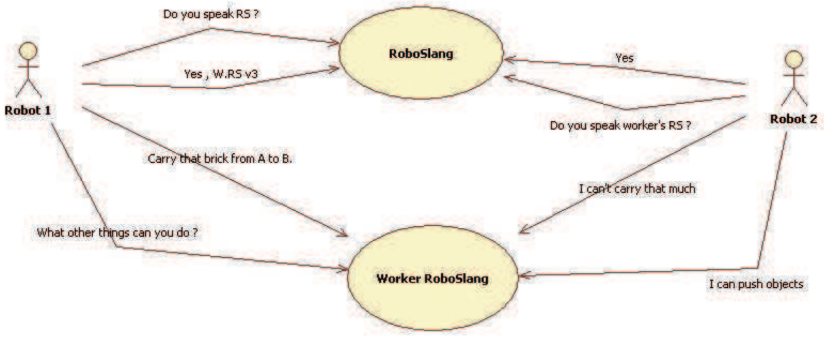


Figure 1. The handshake and dialect choosing mechanism

- Step 1 (Discovery procedure): Robots are inside the “visible“ area of each other. They recognize some hardware signatures (such as wireless signal [3, 4] or an available Bluetooth service [1]) and try to establish a communication channel.
- Step 2 (Handshake mechanism): The handshake procedure consists of exchanging of identification data. This information contains a key (unsigned nonzero integer on 16 bits) that uniquely identifies each device. The uniqueness of the key must be assured by the developer of the platform using a key generator application. In the first version of the system the key will be assign manually at the moment of assigning name to devices, but further improvement may consist of an automatically way of assigning the keys. When a device receives identification data from other devices it must send back a confirmation message. If no confirmation message is received the identification data will be resent. The handshake process ends when each of the devices has received and accepted the identification data of the other devices. Once the communication channel between two devices is set, each of them will assign a local key to the session for the simple fact that between two devices could exist several sessions.
- Step 3 (Dialect choosing): Each robot or device has a list with some known dialects. The basic information about a dialect is:
  - the name of the dialect: 6[1] printable ASCII characters[2];
  - the version of the dialect: 8 bits integer.

[1] There are 94 characters and there 4.2 * 1015 combinations that can be used for choosing the names. The names are given in a fixed size, so if you want to specify a 3 character dialect, then you should put 3 spaces after or before the name.

[2] Printable ASCII characters are numbered from 33 to 126 in decimal notation

More information about the structure of the dialects will be discussed later in this section.

The handshake and dialect choosing mechanism is described in the Figure 1.

After choosing the dialect the two robots will decide what information should change. The structure of the exchange package remains the same for every dialect.

The package structure is as follows:

- 16 bits - the unique identification key of the destination device or 0 for a broadcast. The key is written in fixed 16 bits form.
- 32 bits - representing the length of the data transmitted in the package(number of bytes)
- 1 bit - the checksum bit of an error detection algorithm for the transmitted data the actual data

The proposed checksum algorithm is a simple one revealed by the following function:

$$checksum(x) = XOR_{i=1,..n} x_i, \tag{1}$$

where x_i is the bit representation of x, n is the length of x and XOR is the bits operator applied on multiple bits.

There are some special packages transmitted by the RoboSlang handshake and dialect negotiation protocol. For a better understanding of the concepts of the language and dialects I will use a hex base notation for all the numbers. The first of the series is the presentation package.

It has the following structure:

$$0000h \qquad 000000000000000Fh \qquad checksum(key) \qquad key$$

And the response should be:

$$key \qquad 000000000000000Fh \qquad checksum(key) \qquad key$$

After this procedure the robots will ask each other the interested dialect list:

$$key \qquad 0000000000000040h \qquad checksum(command) \qquad command$$

```
command = {   transmission error == 0h
              retransmit data == 1h
              do you know == 3h dialect version
              yes == 7h
              no == 15h
              begin == 31h
              end == 63h }
```

This codes should be enough for device to establish communication and begin the whole exchange process. The RoboSlang language should be kept as light as possible because it should be used only for initiating the data exchange process.

The codes are chosen to solve the transmission error problem in a easy way, so if you consider the bit representation of these commands you will see that each of it begins with a variable number of zeros followed by a variable number of ones. When receiving a command, a simple check should be made to verify if the data conforms with the specification. If not both error commands (transmission error and retransmit data) will be send.

4. CommonRoboSlang (CRS)

4.1. **The tasks - breaking into small pieces.** CommonRoboSlang is a dialect of RoboSlang and its main goal is to handle the tasks in a uniform manner. Imagine a huge task that a single robot could not handle, but a lot of them can. So you need a proper language to exchange the information about the task and the way it can be divided. Also you would need an algorithm to divide these tasks.

The main principles of CRS are:

- *If you can do a task without any help, do it!*
- *If you can't do a task without any help, then consider the most suitable robot to do it.*

So first of all you would analyze that it is better to split the task and than to do it. A second argument for this should come when there are not any robots available in the area and you need to search them. Also you should consider the transmission time. When all the answers are negative then you should consider splitting as an option.

When you split something you take the task that you can do in an affordable time, but then it comes another question: "How many units of time are affordable¿‘. So CRS answers this question in a simple manner. Try to split the task in as many pieces as the known robots are. Also you should consider the task that you are able to perform.

We talked about splitting tasks into small pieces, but what the tasks are? A task is a collection of abstract steps that you should follow to reach at the end. Abstract steps are known algorithms and they cannot be divided into small pieces. For instance an abstract step can be “go from point x to y“ and also “climb a mountain“. I suggest keeping the steps as small as possible, but if it is necessary you can construct them in a more complex way. These steps are encoded so the transmitted information is as well, as small as possible. The encoding of the steps is chosen by the programmer and it does not have a standard form. If a robot does not know what an abstract step means then it should reject the whole task. The rejection cause can be one of the following: “unknown resource needed“, “unknown steps“ or “task too big“.

The robot should keep a log of the current task so that every time it would know how much time remains.

Nobody should consider that splitting the tasks and describing the abstract steps is an easy job. In fact the intricacy of the problem is equivalent to designing a very complex programming language. The description of the abstract steps is a difficult task and putting them in the task collection can be also very complex. You must decide if to create a task as an abstract step or to split it into small pieces and let the devices decide their flow. In the final stage, when a device is getting and mixing all the results can be also considered a task and could be designed as an abstract step collection.

4.2. **The main commands of CommonRoboSlang.** After switching the conversation to CRS the dialect will have a standard form of question(Q) and answer(A):

Q: status
A: busy means that the other party does not accept any task for the moment
idle this means that the robot is available and ready for any task
none is an undefined state, meaning that the robot have some problems or could not serve any task

Q can you perform [task]
A: yes means that the robot is sure that it can perform the given task
no means that the robot is not able to perform the entire task
possible means that the robot is not sure about the task so it can be rejected later

The other phrases of the dialect have an imperative form:

- job done [task] [result]
- job rejected [cause]

Task assignment should be considered immediately and the results should be given to the robot that assigned the task. Also when all the sub tasks are done the main robot should finish the task also.

5. HumanRoboSlang (HRS)

This dialect is one of final goals of RoboSlang as smart devices should be able to communicate with humans in a natural manner. There are some problems related to this subject and that include human language semantics, natural language processing [11], the criteria to choose a certain language and the list could continue. If we speak of getting the semantics of a certain language we would see that current research in this area is not very surprising [14]. Unfortunately this subject is far away from human dream that we should be able to talk to robots and they would be able to perform tasks given by us. There is some hope also because there are some algorithms capable of answering and learning from humans, but they are still in research and used only for experimental purposes or fun.

The HumanRoboSlang is proposing o way of communicating only certain task to a robot using an SQL like language [15]. So suppose that every task and its description and steps are described in a database. Your only problem now is to figure out what task should be done at a certain time and you take the ID of the task from the TASKS table and insert it into your REQUESTS table. The device will query the table from time to time and perform the tasks ordered by the priority. I know that this is far from a human to robot language, but is a simple way to communicate and it is very easy and low costing solution to implement.

Once the research in the natural language processing is getting some useful results, then a certain solution for HumanRoboSlang should be found, but until then we will be able to communicate with smart devices thought some pseudo natural languages.

6. Implementation

Regarding to the programming languages, there is no restriction for using one or other. If you consider one language better than other, then search for the RoboSlang architecture implemented in your language and begin the programming process. Also there are some cases when you do not really need a RoboSlang implementation and you only want to send or receive some data from a certain language using a known data exchange method.

The platform will be implemented using Java [7] and C on a Linux [5, 6] platform because of the portability and numerous Linux distributions that can be installed on every kind of platform. RoboSlang is not intending for the moment to build a platform of its own, but in time, if it is considered necessary, it would be considered as an option.

As an approach for the platform implementation we can use network programming after we establish an IP connection with a server. The whole scanning process and wireless network [10] detection can be done by the operating system. Another way of implementing the connection between robots is based on bluetooth programming [9]. It can be easily done in Java and Java ME [8] and it does not need any more platforms. There are a few settings to be done under Linux for using bluetooth, but there are a lot of materials on the Internet describing how to do it.

7. Conclusions and further work

The current researches done by some companies suggested that there isn't a certain platform available on the market that could solve the problem of communication between two intelligent devices. So every product comes with its own firmware that it is able to do some tasks that would be designed for, but nobody is concerning in exchanging results and important data between two or more devices. Even if the industrial usage of the robots is extending, the companies propose non distributed solutions for task management and production improve. There is some research for algorithms to control robotic swarms, but there is not a stable platform to test those algorithms. Swarm OS from IRobot Industries is a platform for swarms, but that operating system is still under research, it would not be available for free and their purpose is to control only their hardware (IRobot swarms) [16]. After performing some research you would see that even is a critical problem there are few companies that are implementing some platforms for robots programming, but none of them is dealing with the robots in an autonomous manner and none of them is proposing a distributed solution and a communication platform. RoboSlang will begin as an Open Source project and will became a solution for some of the companies that would like to have real intelligent devices, because they will be able to exchange data and work as a team.

References

[1] Kenneth C. Cheung, Stephen S. Intille, and Kent Larson : An Inexpensive Bluetooth-Based Indoor Positioning Hack, Massachusetts Institute of Technology Press, 2003

[2] Andrew S. Tanenbaum: Computer Networks 4rd Edition, Prentice Hall ,2003

[3] Halk Gmskaya, and Hseyin Hakkoymaz : WiPoD Wireless Positioning System based on 802.11 WLAN Infrastructure, ENFORMATIKA, 2005

[4] Y. Wang, X. Jia, H.K. Lee : An indoors wireless positioning system based on wireless local area network infrastructure, Satellite Navigation and Positioning Group (SNAP),School of Surveying and Spatial Information Systems ,2005

[5] Daniel Bovet , Marco Cesati: Understanding the Linux Kernel, Third Edition, O'Reilly Media, 2005

[6] Linux kernel docs and FAQ `http://www.kernel.org/pub/linux/docs/lkml/`, last visited on 05.05.2008

[7] Sun Java API and Docs `http://java.sun.com/`, last visited on 06.05.2008

[8] Sun Java ME API and References `http://java.sun.com/javame/reference/apis.jsp`, last visited on 06.05.2008

[9] C. Bala Kumar : Bluetooth Application Programming with the Java API , Morgan Kaufmann Publishers, August 2003

[10] Andreas Lerg : Wireless Networks: LAN and Bluetooth, Data Becker, August 2002

[11] Natural language processing
`http://en.wikipedia.org/wiki/Natural_language_processing`,
last visited on 08.05.2008

[12] Hans-Arno Jacobsen: Extensible Middleware, Kluwer Academic Publishers, 2004

[13] Jim Farley : Java Distributed Computing, O'Reilly Media, 1998

[14] I. Heim; A. Kratzer : Natural Language Semantics, Springer Netherlands, 2002

[15] Alex Kriegel, Boris M. Trukhnov : SQL Bible,John Wiley & Sons Inc, 2003

[16] IRobot swarms `http://www.irobot.com/filelibrary/GIvideos/swarm3%20ad.html`, last visited on 12.05.2008

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

E-mail address: so29769@scs.ubbcluj.ro