UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

VANIA BOGORNY
And
ANDREY TIETBÖHL

Extending the Weka Data Mining Toolkit to support Geographic Data Preprocessing

Technical Report – RP 354

Prof. Dr. Luis Otavio Alvares
Advisor

Porto Alegre, July 2006

TABLE OF CONTENTS

LIST OF ABREVIATIONS

KDD	Knowledge Discovery in Databases
KDGD	Knowledge Discovery in Geographic Databases
GDBMS	Geographic Database Management System
DM	Data Mining
SQL	Structure Query Language
CBM	Calculus Based Method
DEM	Dimension Extended Method
GIS	Geographic Information Systems
GDB	Geographic Databases
OGC	Open GIS Consortium
GKB	Geographic Knowledge Base
MBR	Minimum Boundary Rectangle
SAR	Geographic Association Rule Mining
FGP	Frequent Geographic Patterns

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

Large amounts of geographic data have been used more and more in many areas in different application domains such as urban planning, transportation, telecommunication, marketing, etc. These data are stored under Geographic Database Management Systems (GDBMS), and manipulated by Geographic Information Systems (GIS). The latter is the technology which provides a set of operations and functions for geographic data analysis. However, within the large amount of data stored in geographic databases there is implicit, non-trivial and previously unknown knowledge that cannot be discovered by GIS. Specific techniques are necessary to find this kind of knowledge, which is the objective of Knowledge Discovery in Databases (KDD).

KDD is an interactive process which according to (FAYYAD, 1996) consists of five main steps: *selection, preprocessing, transformation, data mining* and *evaluation/interpretation. Selection, preprocessing* and *transformation* are data preparation steps in which data are rearranged to the format required by data mining algorithms. It is stated that between 60 and 80 percent of the time and effort in the whole KDD process is required for data preparation (ADDRIANS, 1996). For geographic databases this problems increases significantly because of the complexity of geographic data that must be considered. We have addressed this problem proposing an interoperable framework for geographic data preprocessing (Bogorny, 2005a) and the use of ontologies to improve the spatial join computation (Bogorny, 2005b).

Data Mining (DM) is the step of applying discovery algorithms that produce an enumeration of patterns over the data. Most of these algorithms were created to deal with small amounts of data and with a restrictive *single table* input format. This limitation causes a *gap* between geographic databases and data mining algorithms.

Many solutions for spatial data mining have been proposed in the literature, but only a few consider data preparation aspects. Most approaches extend query languages with new functions and operations for data mining. Han (1997) proposed a geo mining query language (GMQL) implemented in the GeoMiner software prototype (Han, 1997). Ester (2000) defined a set of new operations such as *get_nGraph, get_neighborhood* and *create_nPaths* to compute geographic neighbors. Sattler (2001) proposed a multi-database language to support the KDD steps. Malerba (2000) proposed an object-oriented data mining query language named SDMOQL, implemented in the INGENS software prototype. In these approaches it is expected that the GDBMS will implement the proposed languages and operations. However, most GDBMS follow the Structured Query

Language (SQL), which became the standard language to manipulate databases, and do not implement data mining languages or operations to automate or semi-automate geographic data preprocessing. Indeed, most geographic data mining software prototypes are no longer available outside academic institutions. Aiming to make a contribution for the spatial data mining field and facilitate the practice of knowledge discovery in geographic databases, we propose an interoperable framework to prepare geographic data for DM. The objective is to automate part of the KDD steps in order to reduce data preparation effort and time. For this purpose we propose to extend the Weka (WITTEN, 2005) data mining toolkit to support geographic data preprocessing. Weka is a well known open source data mining tool which provides friendly graphical user interface for data mining and pattern interpretation.

1.1 Objective and Contributions

The main objective of this work is to automate the geographic data preprocessing steps into the Weka data mining toolkit, aiming to facilitate the practice of spatial data mining. The main contributions include:

- Interoperability with any GDBMS constructed under OpenGIS Simple Features Implementation Specification (OGC 1999a).
- Generate new datasets with more or less information automatically and very easily.
- Different classical data mining algorithms can be applied in the data mining step. The output results can be easily visualized.
- The user can define the relevant spatial features, the target feature type, different spatial relations, as well as different granularity levels.
- Spatial as well as non-spatial data are considered.
- The feature's spatial representation is independent of geometric type. While most spatial data mining algorithms have as an input a set of points, our framework supports any geometric primitive.
- Geographic dependences can be specified and are eliminated between the target feature type and any relevant feature type. This results in more efficient spatial join and the generation of less well known geographic domain patterns, as explained in (Bogorny, 2006a).

1.2 Scope and Outline

This work is limited to provide into the Weka data mining toolkit a module that supports geographic data preprocessing, named GDPM. The remaining of this report is organized as follows: Section 2 presents background knowledge about geographic data. Section 3 presents the extension of Weka, and Section 4 concludes the report and suggests directions of future improvements.

2 BACKGROUND

Geographic databases (GDB) store real world objects, also called spatial features, located in a specific region (OGC, 1999a). Spatial features (e.g. Canada, France) belong to a feature type (e.g. country), and have both non-spatial attributes (e.g. name, population) and spatial attributes (geographic coordinates *x,y*).

The number of geographic feature types in geographic databases is normally large, and every different feature type is usually stored in a different relation, because most geographic databases follow the relational (SHEKHAR, 2003) (RIGAUX, 2000) or object-relational approach. Figure 1 shows an example of geographic data stored in relational databases, where the feature types street, water resource, and gas station are different relations with geographic (shape) and non-geographic attributes.

(a) Street

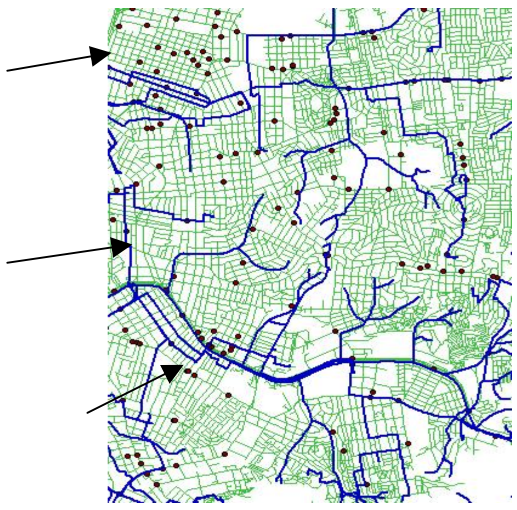
Gid	Name	Shape
1	Erie ST	Multiline [$(x_1,y_1),(x_2,y_2)$,..]
2	Oak ST	Multiline [$(x_1,y_1),(x_2,y_2)$,..]

(b) WaterResource

Gid	Name	Shape
1	Jacui	Multiline [$(x_1,y_1),(x_2,y_2)$,..]
2	Guaiba	Multiline [$(x_1,y_1),(x_2,y_2)$,..]
3	Uruguai	Multiline [$(x_1,y_1),(x_2,y_2)$,..]

(c) GasStation

Gid	Name	VolDiesel	VolGas	Shape
1	BR	20000	85000	Point[(x_1,y_1)]
2	IPF	30000	95000	Point[(x_1,y_1)]
3	Elf	25000	120000	Point[(x_1,y_1)]



(d) GEOMETRY_COLUMNS

F_table_schema	**F_table_name**	**F_geometry_column**	**Type**	**SRID**
Public	Street	The_geom	Multiline	-1
Public	WaterResource	The_geom	Multiline	-1
Public	GasStation	The_geom	Point	-1

Figure 1 – Geographic data storage structure in OGC based GDB

The spatial attributes of geographic object types, represented by *shape* in Figure 1, have intrinsic spatial relationships (e.g. close, far, contains, intersects). Because of these relationships real world entities can affect the behavior of other features in the neighborhood. This makes spatial relationships be the main characteristic of geographic data to be considered for data mining and knowledge discovery (LU, 1993) and the main characteristic which differs geographic/spatial data mining from transactional data mining.

Spatial relationships are usually not explicitly stored in geographic databases, so they have to be computed with geographic operations. There are basically 3 spatial relations to consider (GUTTING, 1994): *topological, distance* and *order*. *Topological* relations characterize the type of intersection between two spatial features and they can be classified in *Equal, Disjoint, Touches, Within, Overlaps, Crosses, Contains* and *Covers*. Figure 2 (a) shows an example of the topological relationships *Touches, Contains* and *Crosses.*

Distance relations are based on the Euclidean distance between two spatial features, as shown in Figure 2(b).

Direction relations deal with the order as spatial features are located in space. Figure 2 (c) shows an example of direction/order relations.

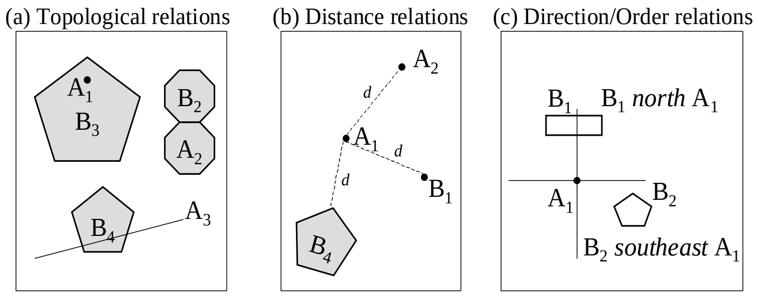


Figure 2 – Spatial relationships

GIS implement specific operations and functions to manipulate and visualize geographic data. The OGC(Open GIS Consortium) is an organization dedicated to develop standards for spatial operations and spatial data integration, aiming to provide interoperability for GIS. Among many specifications established by the OGC, there are two of fundamental importance for this work: operations to compute *spatial relationships* and the *database schema* metadata, introduced in the following section.

2.1 OGC Spatial Operations and Database Schema

The OGC is a not-for-profit organization dedicated to provide interoperability for Geographic Information Systems. It defines a standard set of operations to compute spatial relations for SQL (OGC, 1999b), which are implemented by most GDBMS. These operations are sufficient enough to compute relations among spatial features. Standard functions and operations including *intersection*, *buffer*, *convex hull*, *topological* and *distance* are provided (see (OGC, 1999) to have an overview of the list of operations).

The database schema metadata are stored in a database table named GEOMETRY_COLUMNS, which is created during the database creation and is instantiated automatically when geographic data are loaded the first time. This table stores all database characteristics, including the database schema name, all geographic table names (which is for data mining the geographic feature types), the name of the geometric column, and its type (e.g. point, line). An example is illustrated in Figure 1(d). This table consists of a row for each feature type in the geographic database with geometric attributes.

Figure 3 shows the storage schema where Feature Table/View corresponds to a database table or view, which contains geometric attributes. Each table name is associated with a particular spatial reference system in table Spatial_Reference_Systems. Each geometry in the database is identified by a key (GID), which consists of a collection of elements numbered by an element sequence (ESEQ).

Each Feature Table/View with geometric attributes is related to GEOMETRY_COLUMNS. This table consists of a row for each Feature Table/View with geometric attributes in the geographic database. The data stored for each database table or view with geometry columns include:

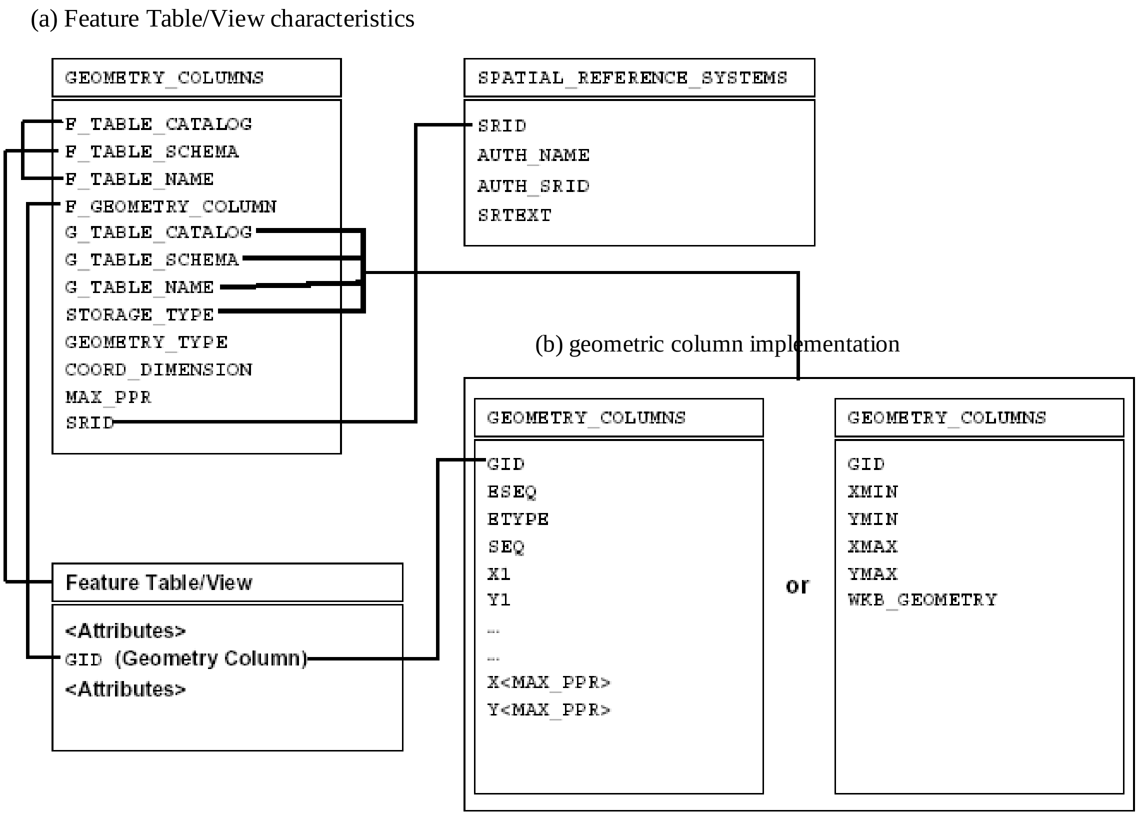


Figure 3 – OpenGIS geographic database schema

- G_TABLE_CATALOG, G_TABLE_SCHEMA, G_TABLE_NAME— name of the geometry table and its schema and catalog. The geometry table implements the geometry column.
- STORAGE_TYPE— type of storage being used for this geometry column.
- GEOMETRY_TYPE—the type of geometry values stored in this column (point, curve,polygon, etc..)
- COORD_DIMENSION—corresponds to the number of dimensions in the spatial reference system.
- MAX_PPR— number of points stored as ordinate columns in the geometry table.
- SRID—the ID of the spatial reference system used for the coordinate geometry in this table. It is a foreign key reference to the SPATIAL_REF_SYS table.

2.2 Query Language and Database Connectivity

The SQL became the standard data definition and manipulation language for relational databases (ELMASRI, 2003). This language is implemented by most commercial and open source GDBMS and was extended with spatial functions and operations to manipulate geographic data. With this standard, it is possible to write queries to access data stored in different databases, without changing the statements.

JDBC is the industry standard for database-independent connectivity between the Java programming language and a wide range of databases. The JDBC API provides a call-level API for SQL database access. With a JDBC API it is possible to establish a connection with a geographic database, to send SQL statements to manipulate data, and to process the results.

3 GDPM: INTO THE WEKA DATA MINING TOOLKIT

Many classical DM algorithms are implemented in toolkits, while spatial data mining toolkits are rare. *Weka*, for example, implements more than twenty different algorithms for classification, clustering and association rules. The algorithms implemented in toolkits share the same single table input format, which can vary a little among different toolkits, but can easily be adapted. When organized into the single table format, different techniques and algorithms can be applied to the same dataset. Figure 4 shows an example of a spatial dataset preprocessed and transformed into the single table format to be mined with Weka.

```
@relation 'geographic_data'

@attribute f_code_des
{null,Cropland,Grassland,Land_Subject_to_Inundation,Rice_Field,Scrub_Brush}
@attribute road_OVERLAPS {yes}
@attribute river_CROSSES {yes}
@attribute river_CONTAINS {yes}
@attribute tunel_CROSSES {yes}
@attribute river_OVERLAPS {yes}
@attribute road_CROSSES {yes}
@attribute bridge_WITHIN {yes}
@attribute river_WITHIN {yes}
@attribute river_TOUCHES {yes}
@attribute bridge_CONTAINS {yes}
@attribute tunel_TOUCHES {yes}
@attribute road_CONTAINS {yes}
@attribute tunel_CONTAINS {yes}
@attribute bridge_OVERLAPS {yes}
@attribute road_TOUCHES {yes}
@attribute tunel_OVERLAPS {yes}
@attribute bridge_CROSSES {yes}
@attribute tunel_WITHIN {yes}
@attribute road_WITHIN {yes}
@attribute bridge_TOUCHES {yes}

@data

Land_Subject_to_Inundation,yes,?,yes,?,yes,?,?,?,?,?,?,yes,yes,?,?,yes,?,?,?,?
Land_Subject_to_Inundation,yes,?,yes,?,yes,?,?,?,?,yes,?,yes,yes,?,?,yes,?,?,?,?
Land_Subject_to_Inundation,yes,?,yes,?,yes,?,?,?,?,yes,?,yes,yes,?,?,yes,?,?,?,?
Land_Subject_to_Inundation,yes,?,yes,?,yes,?,?,?,?,yes,?,yes,yes,?,?,yes,?,?,?,?
Land_Subject_to_Inundation,yes,?,yes,?,yes,?,?,?,?,yes,?,yes,yes,?,?,yes,?,?,?,?
Land_Subject_to_Inundation,yes,?,yes,?,yes,?,?,?,?,?,?,?,yes,?,?,yes,?,?,?,?
Cropland,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,yes,?,?,?,?
Cropland,yes,?,?,?,yes,?,?,?,?,yes,?,?,?,?,?,yes,?,?,?,?
Cropland,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?
Cropland,yes,?,?,?,yes,?,?,?,?,?,?,?,?,?,?,yes,?,?,?,?
Rice_Field,yes,?,yes,?,yes,?,?,?,?,yes,?,?,?,?,?,yes,?,?,yes,?
```

Figure 4 – Weka input format (arff file)

For spatial DM, the single table represents the *target feature type* on which discovery will be performed. Each row in the single table is an independent unit, i.e. a different instance of *target feature type* and each column is an item characterizing this unit. In the example in Figure 4 the target feature type is *city*, such that each row of the single table is a different city. The attributes are spatial predicates related to city.

To generate this file in Weka the user has to open the Weka Explorer interface, shown in Figure 5. This interface has not received any modification.

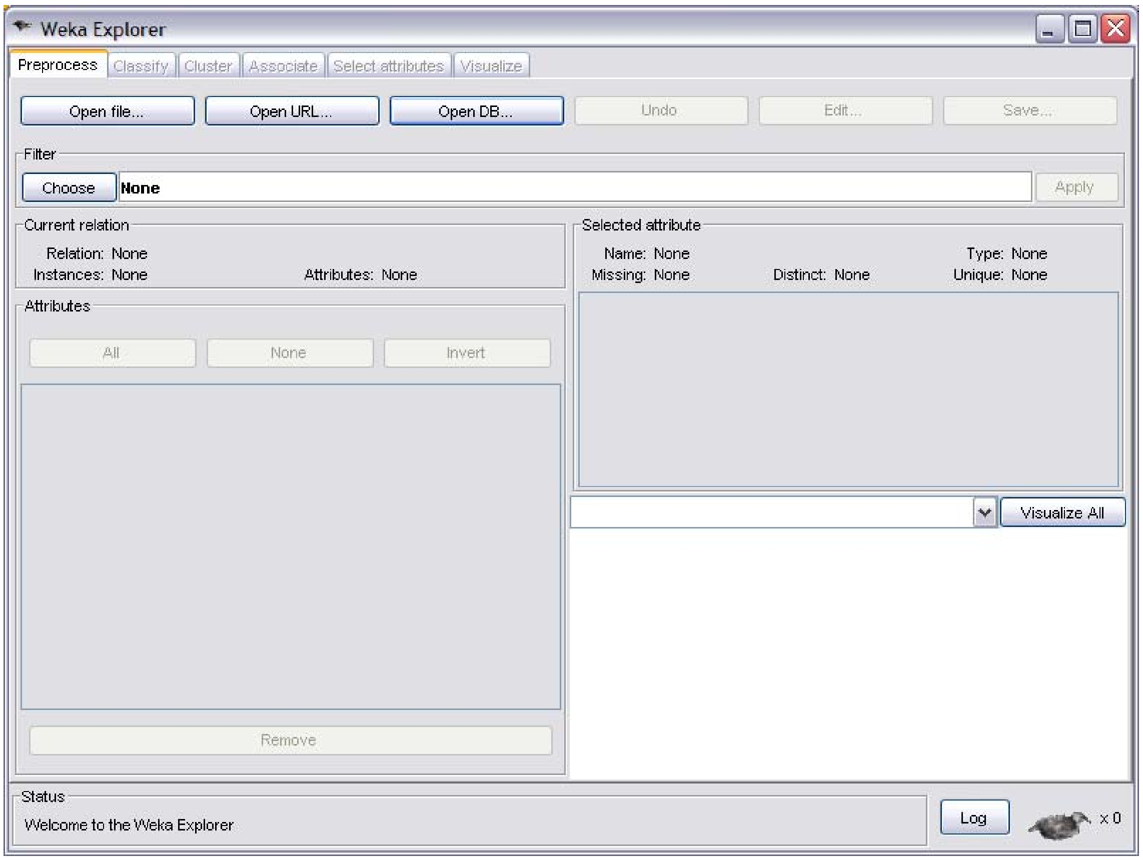
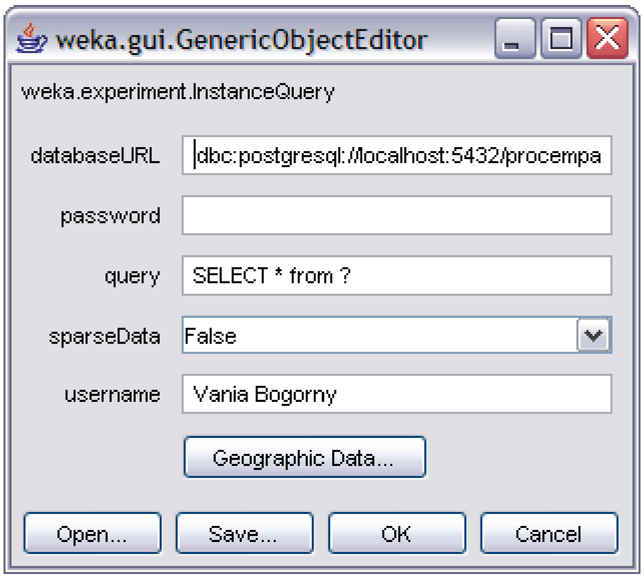


Figure 5 – Weka Explorer GUI

The second step is to click on the button *OpenDB*, which opens the window shown in Figure 6. In the window shown in Figure 6 the user has to provide the database *url*, *user name*, and *password*. This window reads the *DatabaseUtils.props* file to set the default information to connect the database. From this window, Weka connects to PostGIS through JDBC and opens a new window already connected to the database provided by the *url* when the button *Geographic Data (*which we included in order to call the geographic data preprocessing module GDPM) is clicked, as shown in Figure 7.

GDPM is a new java class completely independent, named GeographicData.java, stored into the weka/gui/gdpm directory.


weka.gui.GenericObjectEditor

weka.experiment.InstanceQuery

databaseURL: jdbc:postgresql://localhost:5432/procempa

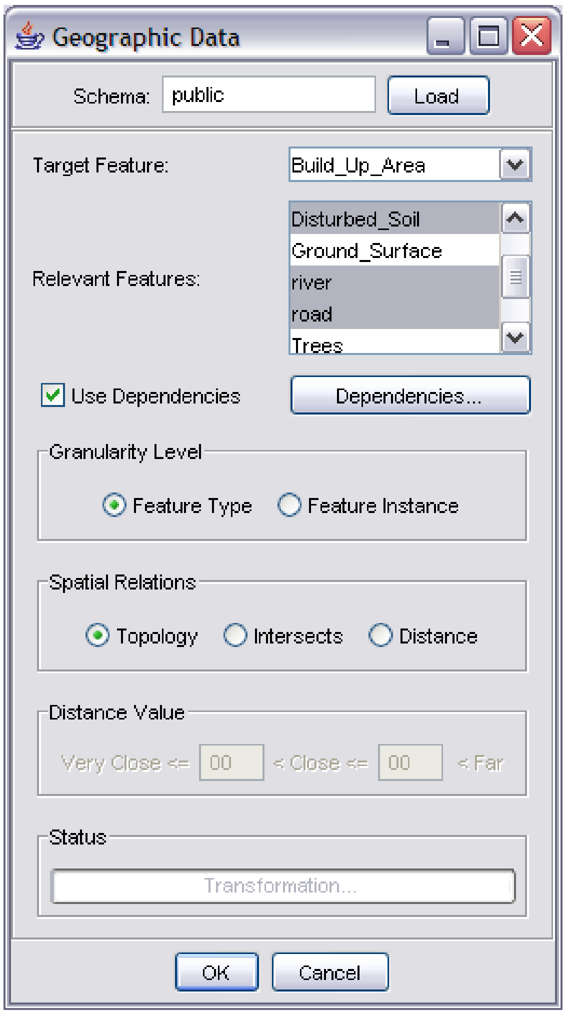
password:

query: SELECT * from ?

sparseData: False

username: Vania Bogorny

Figure 6 – Interface to access geographic databases


Geographic Data

Schema: public

Target Feature: Build_Up_Area

Relevant Features: Disturbed_Soil, Ground_Surface, river, road, Trees

[✓] Use Dependencies

Granularity Level: [✓ Feature Type] Feature Instance

Spatial Relations: [✓ Topology] Intersects Distance

Distance Value: Very Close <= 00 < Close <= 00 < Far

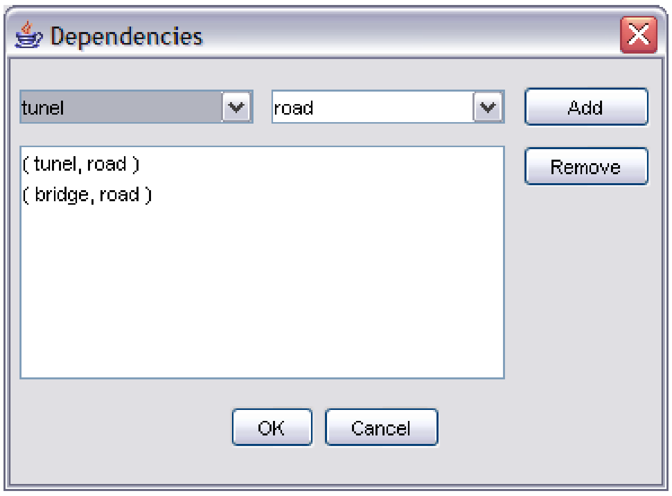
Status: Transformation...

Figure 7 – Geographic data preprocessing interface

To start the preprocessing tasks, the database schema must be provided, and the button *Load* pressed to retrieve all spatial feature types from the database table *geometry_columns*. Since the spatial feature types are loaded, the user chooses the target feature type and all relevant feature types. To define geographic dependences, the user has to mark the check box, which enables the button to specify geographic dependences. After clicking on the *dependence definition button* the interface shown in Figure 8 is called.

In the dependence definition windows appear two combos that contain all database spatial feature types with an instance in the table *geometry_columns*. The user may choose one spatial feature type in each combo, and by pressing the *add* button, a new dependence will be created with the respective selected pair. The pair will than appear in the list of dependences in the same window. To remove any dependence already defined, the user must select the pair and click on the button *remove.*

After dependences have been defined, the button *OK* must be clicked in order to store the dependences into a database table named *knowledgeConstrains.*



(tunel, road)
(bridge, road)

Figure 8 – Geographic dependence definition interface

The following step is to provide the type of spatial relationships and the granularity level in the window shown in Figure 7. Our prototype automatically generates data at two granularity levels (feature instance and feature type), for distance, topological, and high level topological relationships (intersects), without any concept hierarchy, as will be explained in the following section. For distance relationships one or two distance parameters must be provided.

Since all parameters have been provided to the window shown in Figure 7 and button *OK* is pressed, the first step performed is the dependence elimination. Before perform the spatial join step between the target feature type and each relevant feature type, GDPM searches for a dependence in the table *knowledgeConstraints.* When a dependence is found, the relevant feature type is removed from the set and the next relevant feature type is tested. After all dependences have been removed the spatial join and transformation modules start.

3.1.1 The Spatial Join and Transformation Process

3.1.1.1 Topological Relationships

Topological relationships are mutually exclusive, i.e., only one topological relationship holds between to spatial feature *instances* (e.g. Porto Alegre city and Canoas city). At the feature instance granularity level, i.e., when both type and instances of the relevant feature types are considered in the mining process, every instance of the target feature type may have one and only one topological relationship with an instance of a relevant feature type. Table 1 (left) illustrates an example of the spatial join computation where city 1 has the relationship *contains* with River_1, *crosses* with River_2, and *contains* with Slum_1. The spatial join output (Table 1 left) at the feature instance granularity level when transformed to the Weka input format (Table 1 right), the relevant feature type name with the respective instance is transformed in an attribute, while its value receives the respective topological relationship. For the relevant feature instances that have no relationship with the instance of the target feature (e.g. River_3 and city_1, River_1 and city_3), the attribute value is filled with "?", which is the symbol used by Weka to represent the absence of an attribute.

Table 1 - *Feature Instance* Granularity Level for topological relationships

TargetF_id (city)	RelevantF Instance	Relationship
1	River_1	Contains
1	River_2	Crosses
2	River_3	Contains
2	River_4	Crosses
3	River_2	Crosses
1	Slum_1	Contains
2	Slum_2	Contains

TargetF_id (city)	**River_1**	**River_2**	**River_3**	**River_4**	**Slum_1….**
1	contains	crosses	?	?	Contains
2	?	?	Contains	Crosses	Contains
3	?	Crosses	?	?	?

At a higher granularity level (feature type), as shown in Table 2 (left), notice that the relevant feature instance is not stored in the spatial join output. For example, city 1 has two topological relationships with the relevant feature type River, *contains* and *crosses.* At this granularity, to preserve the type of the topological relationship, we cannot simply convert the relevant feature type River to an attribute in the transformation process because attributes with same names are not allowed. For example, the final table cannot have two attributes River with two different relationships (contains and crosses). Indeed, it would be difficult to specify dominance between topological relationships in order to choose the strongest relationship.

To preserve the relationship type, the attribute name in the final table (Table 2 right) is the feature type and the relationship type, while the attribute value receives the string "yes" when the relationship holds and " ?" if there is no topological relationship.

Table 2 - *Feature Type* Granularity Level for topological relationships

TargetF_id (city)	RelevantF Type	Relationship
1	River	Contains
1	River	Crosses
2	River	Contains
2	River	Crosses
3	River	Crosses
1	Slum	Contains
2	Slum	Contains

TargetF_id (city)	**Contains_River**	**Crosses_River**	**Contains_Slum**
1	Yes	Yes	Yes
2	Yes	Yes	Yes
3	?	Yes	?

According to the objective of the discovery sometimes topological relationships may generate some problems. Let us evaluate these problems in practice analyzing Figure 9 which shows part of a geographic map where the small polygons are slums, large polygons are districts, and lines are water bodies of the city of Porto Alegre. Let us suppose that district is the target feature type and slums and water bodies are the relevant feature types.

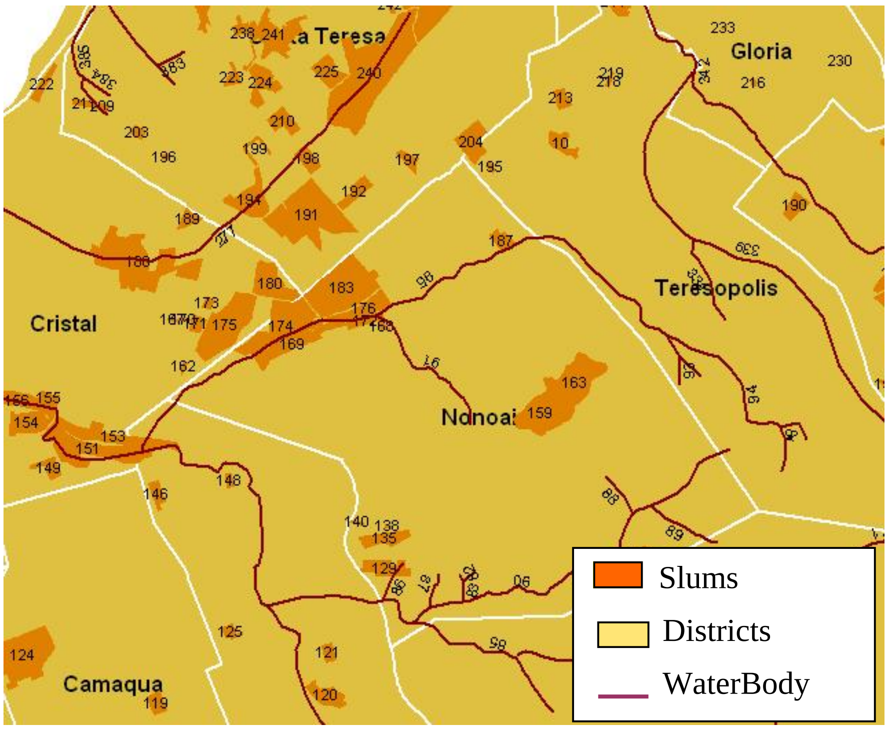


Figure 9 – Part of a geographic map of the Porto Alegre city representing districts, slums, and water bodies

In Figure 9 we can observe the different topological relationships that districts may have with both slums and water bodies. The district *Nonoai*, for example, "*contains*" slums (e.g. 159 and 183), "*touches*" slums (e.g. 180), and "*overlaps*" slums (e.g. 174). The district *Teresopolis*, for example, "*contains*" water bodies (e.g. 93, 338) and is "*crossed*" by water body (e.g. 339). Different topological relationships may not generate patterns when mining data at the feature *instance* granularity level if thresholds such as minimum support for mining spatial association rules are not very low.

For example, the predicates *touches(slum_183)* for district Santa Tereza will be considered different from the predicate *contains(slum_183)* for the district Nonoai. The same occurs for slum_180 which is *within* district Cristal and *touched* by district Nonoai. When the objective is to investigate high criminal incidence, for example, and either slum 180 or 183 are responsible for high criminal incidence in districts Nonoai and Cristal, this might not be discovered by data mining algorithms. The same problem may occur when mining data at the feature type granularity level.

As one solution for this problem we propose to consider general topological relationships intersects and non-intersects. When different instances of the target feature (e.g. district Cristal and district Nonoai, or district Nonoai and district Santa Teresa) have topological relationships with the same instance of a relevant feature type (e.g.*slum_180* and *slum_183*), instead of *touches(slum_180)* and *contains(slum_180)*, for example, a predicate *intersects(slum_180)* is generated.

Table 3 (left) shows two tables with the feature instance (top) and feature type granularity level (bottom) for the intersects relationship, and Table 3 (right) shows the respective transformation for the Weka input format.

Table 3 – (left) Feature instance and feature type granularity for high level topological relationships (intersects and non-intersects) and (right) Weka input format

TargetF_id (city)	RelevantF Instance	Relationship
1	River_1	Intersects
1	River_2	Intersects
2	River_3	Intersects
2	River_4	Intersects
3	River_2	Intersects
1	Slum_1	Intersects
2	Slum_2	Intersects

TargetF_id (city)	**River_1**	**River_2**	**River_3**	**River_4**	**Slum_1….**
1	intersects	intersects	?	?	intersects
2	?	?	intersects	intersects	intersects
3	?	intersects	?	?	?

TargetF_id (city)	RelevantF Type	Relationship
1	River	Intersects
2	River	Intersects
3	River	Intersects
1	Slum	Intersects
2	Slum	Intersects

TargetF_id (city)	**Intersects_River**	**Intersects_Slum**
1	Yes	Yes
2	Yes	Yes
3	yes	?

3.1.1.2 Distance Relationships

Distance relationships are computed according to the distance parameters provided by the user. If only one distance parameter is provided, neighborhoods are considered very close if their distance from the target feature is less or equal to *dist1*. When two distance measures are informed, than neighborhoods are considered *very close* if their distance from the target feature is less or equal to *dist1* and *close* if their distance is higher than *dist1* and smaller than *dist2*, as shown in the example in Table 4 (left) for the feature instance and feature type granularity level.

It is important to emphasize that GDPM does not materialize the *far* relationship, because all relevant feature types that are neither close nor very close from the target feature will be far. This would produce and enormous amount of non-interesting patterns. For example, the city 1 is very close to river 1 and close to river 2, but far from all other rivers. The materialization of the relationship *far* would generate, for example, patterns for all relevant feature types *far* from the target feature. For distance relationships we can say that close relationships are dominant over far, because all objects are *not close* will be far. Examples of patterns generated at lower granularities include as *River_1=VeryClose →* *River_3=far*, and contradictory/negative rules at higher granularities, such as *VeryClose_River=yes →FarFrom_River=yes*.

To better understand this process, let us observe the map shown in Figure 9. Slum 159, for example, is very close to district Nonoai, but it would be far from all other districts for very low distance threshold. At the feature type granularity level, the district Nonoai would be *very close* to slums (that are within Nonoai), *close* to slums (that are within districts that share boundaries with Nonoai such as Cristal, Santa Tereza, and Teresopolis), but *far* from *all* other slums related to all other districts (e.g. slum 124 in district Camaqua) far from Nonoai which do not appear in the map.

Table 4 - (left) Feature instance and feature type granularity for high level for distance relationships and (right) Weka input format

TargetF_id (city)	RelevantF Instance	Relationship
1	River_1	VeryClose
1	River_2	Close
2	River_3	Close
2	River_4	Close
3	River_2	VeryClose
1	Slum_1	Close
2	Slum_2	VeryClose

TargetF_id (city)	**River_1**	**River_2**	**River_3**	**River_4**	**Slum_1….**
1	VeryClose	Close	?	?	Close
2	?	?	Close	Close	VeryClose
3	?	VeryClose	?	?	?

TargetF_id (city)	RelevantF Type	Relationship
1	River	VeryClose
1	River	Close
2	River	Close
3	River	VeryClose
1	Slum	Close
2	Slum	VeryClose

TargetF_id (city)	**VeryClose_River**	**Close_River**	**Close_Slum**	**VeryClose_Slum**
1	Yes	?	Yes	?
2	?	Yes	?	Yes
3	Yes	?	?	?

Just in case someone would like to consider things that are *far*, the value of the distance metric *dist2* can be increased in order to cover the relationship far.

In the previous sections we described the details to be considered for preprocessing geographic data. In the following sections we describe implementation details.

3.1.2 Implementation details

3.1.2.1 Spatial Join

The spatial join step is performed among the target feature type and all selected relevant feature types that have no dependence with the target feature. This is performed in the geographic database, with the spatial operations implemented by PostGIS, which follows the OGC approach.

The spatial join result is stored in a database temporary table called *target_feature_type_name_temp.* This table contains the attributes *gid_target_feature_type, relevantF* (which is the name of the relevant feature type), and *relationship,* as described in the previous sections.

In case there are no spatial relationships between the target feature type and relevant feature types, the window shown in Figure 10 is presented to the user.

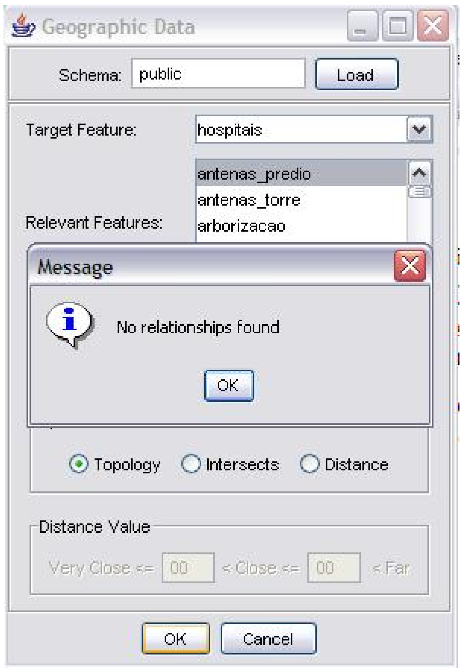


Figure 10 – Error message when no relationships are found

3.1.2.2 Transformation

The transformation module is performed in memory and generates an *arff* [1]file in the directory weka/data, named *geographic_data.arff*. As the *arff* file may contain a large number of attributes, mainly when mining data at the feature instance granularity level, the transformation step cannot store the result in a database table, since PostGIS only allows database tables with less than 1500 columns.

By implementing the transformation step in memory makes the process run much faster. Going into detail about the implementation, the transformation step implies in obtain the non-spatial attributes of the target feature type (obtained from the database table target_feature_type), and the spatial predicates, which are all different predicates generated by the spatial join step.

The transformation step starts reading from the database the table *target_feature_type_temp*, in order to find the spatial attributes (spatial predicates) that will be part of the header of the *arff* file. To find the new attributes the system checks which granularity level is selected in the interface. At the feature instance granularity level, every different value of the column *RelationF* in table *target_feature_type_temp* will result in a new attribute in the *arff* file.

The possible values of these attribute are stored in the column *relationships* in the table *target_feature_type_name_temp*. For topological relationships, possible values of the *relevantF* collumn are: "CONTAINS", "TOUCHES", "WITHIN", "OVERLAPS", "CROSSES", as shown in the example in Table 1. For the high level topological relationships, the possible value is "INTERSECTS", as shown in Table 3. For distance relationships the possible attribute values are "VERY_CLOSE" and "CLOSE".

[1] .arff is the input format required by Weka.

At the feature type granularity level, where relevant feature types may have different topological relationships with the target feature type, the attribute names in the *arff* file will be the different values of the column *relevantF* concatenated with the value of the column *relationship* stored in the temporary table *target_feature_type_temp.* For the feature type granularity level, the values of the attributes in the matrix will be "YES" when the relationship holds in the respective value of column *relationship* in the table *target_feature_type_temp*.

After discovering the attribute names that will be the header of the *arff* file, a matrix is created. There will be exactly one row in the matrix for every different instance of the target feature type, i.e., for every different *gid* (geographic identifier). The columns of the matrix will be the attribute names discovered in the previous step. The matrix is initialized with "?" strings, to fill the attribute values if there is no relationship between an instance of the target feature type and a relevant feature (instance or type).

To filling the matrix with the respective relationships, the temporary table is scanned again. Then the matrix is updated with all records that have a relationship (different from "?"). Having the matrix matched the temporary table, finally the *arff* file will be created.

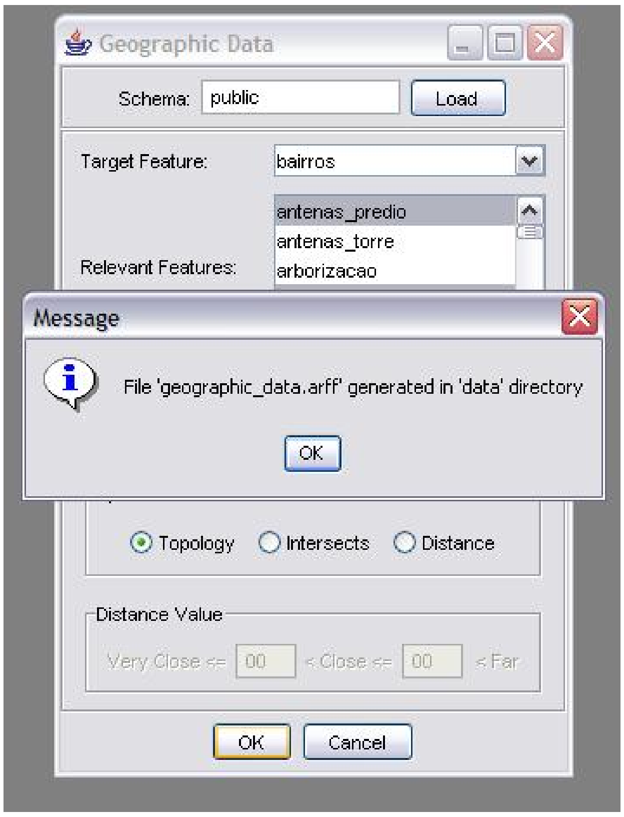
To create the *arff* file GDPM starts looking for the non-spatial attributes of the target feature type in the database, saving their names into a vector, named *columns*. The header of the file I then created writing each column in *columns* as an attribute name. The type to be created in the *arff* file depends on the respective attribute type in the database. For the type *string* we need to select all distinct values that the attribute may have in order to create the header of a string attribute with all its possible values. For this purpose a SQL query is performed over the string attributes of the target feature type. After that, GDPM enumerates these values in the *arff* file, additionally including the value *null* to catch the null values in the database.

Since non-spatial attributes have already been created, GDPM writes the spatial predicates (which are columns in the matrix generated above) as attribute names in the *arff* file. In the sequence, the different values that each of this attributes may have is written in an enumeration of values.

To filling the body of the file (*@data*), GDPM starts executing a SQL query that returns all instances of the target feature type with its respective non-spatial attribute values. For every instance, the respective non-spatial attribute values returned by the query as well as the spatial predicates (in the matrix) are added to the file.

In order to avoid errors in the *arff* file all characters that are not in the sets [A-Z,a-z,0-9]), are replaced in both attributes and values to the character '_'.

Since the *arff* file is created, a message is shown to the user, as shown in Figure 11.



File 'geographic_data.arff' generated in 'data' directory

Figure 11 – Finishing message when the arff file is successfully created

3.1.2.3 Database auxiliary functions to improve performance

Spatial join is the processing bottleneck in spatial data analysis and knowledge discovery. In order to improve the spatial join computation aiming to make spatial queries both easier and faster, some auxiliar database functions were created. These functions are written in the plpgsql language, and are described in the *auxFunctions.txt*, that have to be placed into the Weka installation directory.

Weka tries to create these functions automatically in a PostGreSQL database after clicking the *geographic data* button shown in Figure 6. The functions are very simple and are basically a *switch* that selects the appropriate label given two geometric attributes. The *switchIntersection* chooses the suitable topological relationship. The *fHowDistance* determines among "CLOSE" and "VERY_CLOSE" given two geometric attributes and two distance parameters.

3.1.2.4 Weka modifications

The Weka Toolkit was projected to work with JavaBeans. As a consequence, the correct point to insert the new code is not a trivial task. The *PropertySheetPanel* class is responsible to create the panels with their respective properties. One of these panels is the one shown in Figure 6, where the new button called *Geographic Data* was added.

Weka creates all instances dynamically at the beginning when it is loaded, calling the *PropertySheetPanel* class for each instance. To create the new button only in the specific panel called *weka.Explorer*, in the *DataPreprocessing* interface, we added an *if* command that asks for an *InstanceQuery* object. The *InstanceQuery* is the bean used to mount the panel, and it contains all information necessary to connect to the database. When the *InstanceQuery* is found, the new button is created.

Additionally, it needed to know the *InstanceQuery* object, because the new module GDPM (Geographic Database Preprocessing Module) need to get the information about the user´s connection of this object. Thus, a listener adapter inner class was created to mantain this object. The adapter class calls the GDPM module with the information of its InstanceQuery object. The adapter class is at the end of the file *PropertySheetPanel*.java, and is named *instanceQueryAdapter*.

3.1.2.5 New classes and Methods

A new class named *GeographicData.java* is created into the *weka/data/gdpm* directory. It is independent and does not affect any other class or the Weka original code. The main methods of this class are: topology, intersects, distance, and transformation, which respectively implement all aspects mentioned before.

The dependence elimination is implemented in another class, named Dependences.

Implementation details my be found in Apendix 1.

4 CONCLUSIONS AND FUTURE WORKS

This work presented a solution for the problem of automatic geographic data preprocessing addressed in (Bogorny, 2005a) and (Bogorny, 2005b). The dependence elimination between the target feature type and relevant feature types reported in (Bogorny, 2006a) has also been addressed and solved into GDPM 1.1.

The next steps include the implementation of the algorithms Apriori-KC (Bogorny, 2006b) and Max-FGP (Bogorny, 2006c) in Weka to eliminate geographic dependences among relevant feature types.

5 REFERENCES

(a) BOGORNY, V.; ENGEL, P. M.; ALVARES, L.O.: A Reuse-Based Spatial Data Preparation Framework for Data Mining. In Proceedings of the 17 th International Conference on Software Engineering and Knowledge Engineering, (SEKE'05). 14-16 July, 2005. Taipei, Taiwan, Republic of China, 2005, 649-652.

(b) BOGORNY, V.; ENGEL, P. M.; ALVARES, L.O.: Towards the Reduction of Spatial Join for Knowledge Discovery in Geographic Databases using Geo-Ontologies and Spatial Integrity Constraints. In: Second Workshop on Knowledge Discovery and Ontologies (KDO'2005), Porto, Portugal, 2005, 51-58.

(a) BOGORNY, V.; CAMARGO, S.; ENGEL, P. M.; ALVARES, L.O. Towards elimination or well known geographic domain patterns in spatial association rule mining. In: Proceedings of the 3th IEEE (IS'2006) International Conference on Intelligent Systems, London, September 4-6, 2006 (to appear).

(b) BOGORNY, V.; ENGEL, P. M.; ALVARES, L.O. GeoARM – an interoperable framework to improve geographic data preprocessing and spatial association rule mining. In: Proceedings of the 18th SEKE (SEKE'2006) International Conference on Software Engineering and Knowledge Engineering.

ADRIAANS, P. AND ZANTINGE, D (1996). *Data mining.* Addison Wesley Longman, Harlow, England.

ESTER M, KRIEGEL H-P, SANDER J (1997). Geographic Data Mining: A Database Approach. *In Proceedings 5th Int. Symposium on Large Geographic Databases (SSD*), Berlin, Germany, pp. 47-66.

ELMASRI R, NAVATHE S (2003). Fundamentals of Database Systems. (4) Addison-Wesley.

FAYYAD U, PIATETSKY-SHAPIRO G AND SMYTH, P (1996). From data mining to discovery knowledge in databases**.** *AI Magazine*, 3(17): 37-54.

GUTING, R. H. An Introduction to Spatial Database Systems. **The International Journal on Very Large Data Bases**, V3 (4), (October) , pp. 357 - 399 (1994)

HAN J, KOPERSKI K., STEFANVIC N (1997) GeoMiner: a system prototype for geographic data mining. In *Proceedings of the ACM-SIGMOD international conference on*

Management Of Data (SIGMOD'97) (May 13-15,1997). ACM Press, Tucson, AR, 553-556.

LU, W.; HAN, J.; OOI, B. C. Discovery of general knowledge in large spatial databases. In Proceedings… Far East Workshop on Geographic Information Systems, 275-289, Singapura, 1993.

MALERBA D, ESPOSITO F, LANZA A, ET AL (2000). Discovering geographic knowledge: the INGENS system. In *Foundations of Intelligent Systems, 12th International Symposium, (ISMIS)*, Lecture Notes in Artificial Intelligence, 1932, 40-48, Springer, Berlin, Germany.

OGC (1999a). Topic 5, the OpenGIS abstract specification – OpenGIS features – Version 4. Available at <http://www.OpenGIS.org/techno/specs.htm>. Retrieved August 2005.

OGC (1999b). OpenGIS simple features specification for SQL. In URL: http://www.opengeogeographic.org/docs/99-054.pdf

RIGAUX, P.; SCHOLL, M., VOISARD, A. **Spatial Databases: With Application To GIS**. San Francisco: Morgan Kaufmann Publishers, 2002.

SATTLER K, SCHALLEHN E (2001). A Data Preparation Framework based on a Multidatabase Language. In Proceedings of International Database Engineering and Applications Symposium (IDEAS).

SHEKHAR, S., CHAWLA, S. **Spatial databases: a tour.** Prentice Hall, Upper Saddle River, NJ, 2003.

IAN H. WITTEN AND EIBE FRANK (2005) "Data Mining: Practical machine learning tools and techniques", 2nd Edition, Morgan Kaufmann, San Francisco, 2005.