Creating Standalone Spatially-Enabled Python Applications Using the ArcGIS Geoprocessor

Laura C. Rodman
John Jackson

Abstract

With the introduction of ArcGIS 9 and Python support, it is now possible to write standalone applications in Python that use the geoprocessing functionality in ArcGIS. This provides an option to VBA, C++, or Java for developing spatial applications. Programming in Python is fast and allows for efficient development as well as connectivity to a vast number of open source programs written in Python, C, or C++. A Python application is described which performs data mining in geographic data sets. This program discovers association rules that describe the relationships between geographic objects. This software relies on the ArcGIS geoprocessor in combination with Python open source libraries. For example, the analysis requires a relational database. The Python application can use either the ESRI Personal Geodatabase or an external open source database such as MySQL. The development of this Python application, including packaging and GUI issues, will be discussed.

Introduction

A spatial data mining application is being developed for the U.S. Army Corps of Engineers Topographic Engineering Center [1]. This application can be used to discover association rules that describe relationships between geographic features. Knowledge of terrain, obstacles, travel routes, buildings and structures, vegetation, and land and water features is essential, to improve situational and environmental awareness and to support decision making. Although much of this information is stored in geographic data sets, relationships or patterns among multiple objects or attributes may not be quickly and easily understood, due to the sheer volume of data and the need for a comprehensive analysis. There may also be a lack of data in some geographic locations. In those situations, the ability to infer the presence of features in an area based on similar patterns of occurrence elsewhere would be of use to Army decision makers. Association rules can also apply to the interpretation of remote sensing images and can provide guidance for data set validation and error checking.

The association rule data mining application, called *Aspect*, works in conjunction with ArcGIS, relying on its geographic data formats, coordinate system conversion, mapping, and geoprocessing technology. After ESRI announced support for Python in ArcGIS version 9.0, the decision was made to develop *Aspect* in Python. Python is a powerful open-source programming language. Programming in Python is relatively easy with a short learning curve, thus making the project development easier and more fun. The advantages to Python are numerous. It is cross-platform, so Python scripts (including graphical user interfaces) can run on Windows, Macintosh, or Linux/Unix. The language interpreter is small enough that it can be included within an executable, so there are no concerns about distributing a Python

application to users who might not have the appropriate version available on their machines. Python code is designed to be readable, and since it is open source, it is easy to work with other people's code. This is important, since there is a wealth of Python scripts freely available on the Internet that accomplishes many tasks. The libraries and dependencies present in Python scripts are usually straightforward, and since the source code is often available, it is possible to fix bugs or extend the functionality of other people's code. In contrast, using someone else's C++ code can be problematic, since there may be missing libraries, version incompatibility, and difficulties in resolving the code dependencies and understanding different coding styles.

Available Python libraries include tools to connect to commonly used relational databases, and a comprehensive GUI (Graphical User Interface) widget library. Importantly for a spatial application, the geoprocessing capabilities in ArcGIS have become available through Python. The lessons learned while developing the data mining application, in particular how to make a standalone Python application that is spatially enabled using the ArcGIS geoprocessor, will be described in this paper.

Application Requirements

The software development decisions were guided by the requirements of the application. Two major components of *Aspect* are a relational database management system and a graphical user interface. These two components and how they were integrated into the application using Python are discussed in detail in the following sections.

The geoprocessing functionality is used extensively in *Aspect*. The input data for the analysis are vector data sets that are in shapefile or feature class format. Shapefiles are read using pure Python; however, to read feature classes, *Aspect* connects to the ESRI personal geodatabase and uses its methods to read in shapes. For some operations feature classes are also created, thus functions to write feature classes are used. Samples are collected throughout the domain using various sampling schemes based on the data shape types. The sampling uses geoprocessing tools such as intersect and clipping, as does the internal analyses for handling multivariate association rules.

Relational Database Management System

A relational database is a critical part of the data mining application. To perform the analysis, data is read in from geographic data files, either shapefiles or feature classes. This data must then be converted to a format that can be used to discover associations. Each vector shape from the input data must be tracked, along with its relevant attribute name/value pairs and spatial locations. Spatial objects or attribute values are grouped together in various ways depending on the associations of interest. By rewriting the original data into a database, the dependence on the original file format is removed, and complex SQL queries for the different objects or attribute values can be performed in a standardized fashion.

Aspect provides two choices for the relational database. The first choice is to use the personal geodatabase that comes with ArcGIS. An analysis geodatabase is created (or an

existing one is used), and tables are created in that geodatabase to handle the data mining operations. The advantage of using the personal geodatabase is that it comes with ArcGIS, and the user does not need to install, configure, or manage an external relational database management product. This makes the data mining analysis easier for users, and provides tighter integration with ArcGIS. Implementation in Python is easy, as Python calls to the ArcGIS geoprocessor allow for SQL operations to be performed on the geodatabase. However, the disadvantages to the geodatabase are speed and SQL limitations. SQL queries go through several layers (geoprocessor object calls to geodatabase to underlying database engine), which slows down the operations.

A second choice is to use an external relational database. The current version of *Aspect* provides an implementation using MySQL, which is a popular open-source database management system [2]. A library for Python to communicate with MySQL databases is commonly available. SQL queries are easily performed in Python to perform database operations, they are much faster than the same queries made against the ArcGIS geodatabase, and they can be more complex.

Some important points of database programming and lessons learned are summarized here.

(1) Modularization of the application software is a good practice, so that different database products can be easily swapped in and out of the application. The data mining application is not inherently limited to either the personal geodatabase or to MySQL. A multiuser geodatabase or another external database product (Oracle, DB2, etc.) can easily be used instead. Since Python is an object-oriented language, it is easy to write code that is modular.

(2) The personal geodatabase is built upon the Microsoft Jet Engine (MS Access database product). To improve computational speed, it is possible to use Python to directly access this underlying database, thus bypassing the outer layers that slow down the operations. It is critical that any direct access to the database be limited to "read-only" queries to avoid corrupting the geodatabase. Any "write" operations must be performed through the ArcGIS geoprocessor object. Note that a speedup using direct access might not be realized if there is a sequence of read and write operations. It is time-consuming to switch back and forth between direct access and the geoprocessor operations; in that case, it is more efficient to use the geoprocessor only.

If direct access is to be used, it is necessary to close the direct connection to the database before switching to ArcGIS geoprocessing commands to write to the database or to modify the database schema. This is because ArcGIS requires an exclusive lock on the database when it performs database operations.

(3) In some circumstances direct access to the Microsoft Jet Engine is necessary, not for speed, but because the geodatabase limits the types of SQL queries that are possible. For example, there is only limited functionality of SQL "where" clauses, and there is no "order by" clause. This functionality is not included in ArcGIS since these queries are rare for

geographic objects in feature classes. However, the data mining analysis goes beyond geographic objects and uses a variety of data types within standard database tables in the geodatabase. For complex SQL queries on these tables, direct access may be required.

(4) A fast alternative to performing complicated database queries is to read the data into memory and manipulate them in Python instead. Certain operations, such as sorting, are optimized in Python, and it is easier and faster (especially during development) to perform these operations in Python rather than to use a database. This is another alternative for avoiding the SQL query limitations in the geodatabase.

(5) To speed up the import of data into the database (for example, while loading shapes from feature classes or shapefiles into the database), it is helpful to cache the "insert cursors" for each of the tables as they are needed. The cursors are then deleted at the end of the transaction. By reusing the cursors (in a single transaction) rather than creating them and deleting them, the data operations can be sped up.

Graphical User Interface

A second critical component to a full-function application is a graphical user interface (GUI). For the data mining application, the GUI includes graphical elements beyond buttons and menus, such as hierarchical tree controls, event notification between multiple windows, HTML table displays, and more.

The GUI is written using wxPython [3], which is a blend of Python with the wxWidgets C++ library. wxWidgets is a freely available library, and its cost-free license has generous terms for reuse (fewer constraints than the GNU Public License). wxWidgets has advantages over the other major open-source GUI toolkit, Tk. First, since wxPython is written in Python, it integrates easily with a Python application. Second, wxWidgets uses the underlying native interface where possible, while Tk does not. The result is a visual appearance that looks native for whatever platform it is running on, so it appears to be Windows XP on Microsoft Windows, Mac OS X on Macintosh, etc. Since Python is cross-platform, the same GUI will run on any platform. Windows-only tools are available for GUI development, such as Visual Basic; however, these tools aren't free and have steeper learning curves. wxPython has a large user base maintaining it, so there is confidence that wxPython will be ported to future hardware platforms.

The tree control widget provides an example of the advantages of wxPython. The tree control included in wxPython was written in C++, but it was missing some functionality that was needed for the data mining application. Also, the implementation had a few bugs. Unfortunately, it is difficult to fix bugs in C++. However, someone posted a pure Python implementation of a tree control on the Internet. The Python version also had bugs, but they were easier to fix than the C++ version. It was also easy to see how the code was written and to extend the functionality. For example, the ability to do multiple selection drag-and-drop was added to the tree control. The tree control now works very well in the data mining GUI.

Figure 1 shows the hierarchical tree control. *Aspect* uses a hierarchy of geographic objects, and the association rules are formed from members of any level of the hierarchy. The top level consists of the data layers or geographic object names. The middle level consists of the attribute names associated with the geographic objects (the user has previously selected only those attributes that are relevant to the analysis), and the bottom level consists of the attribute values. The hierarchical tree is shown in the left-most box. The tree control allows the user to group some of the attribute values together into more general categories, which is necessary if there are infrequent occurrences of some values. The data mining analysis presupposes that there are large frequencies of occurrence of the data samples, so if an attribute value occurs infrequently, it may be combined with other values to make a more general variable. (The results of a contingency table analysis indicate to the user which variables have too few samples and should be combined.) In Figure 1, the vegetation types "AGS" (Annual Grass) and "PGS" (Perennial Grass) are combined into a more general category "Grassland." The ability to create these bins and to group items into them is more sophisticated than the capabilities of the basic tree control in wxWidgets. Fortunately the extensibility of Python and the availability of shared code on the Internet allowed this extra capability to be added.

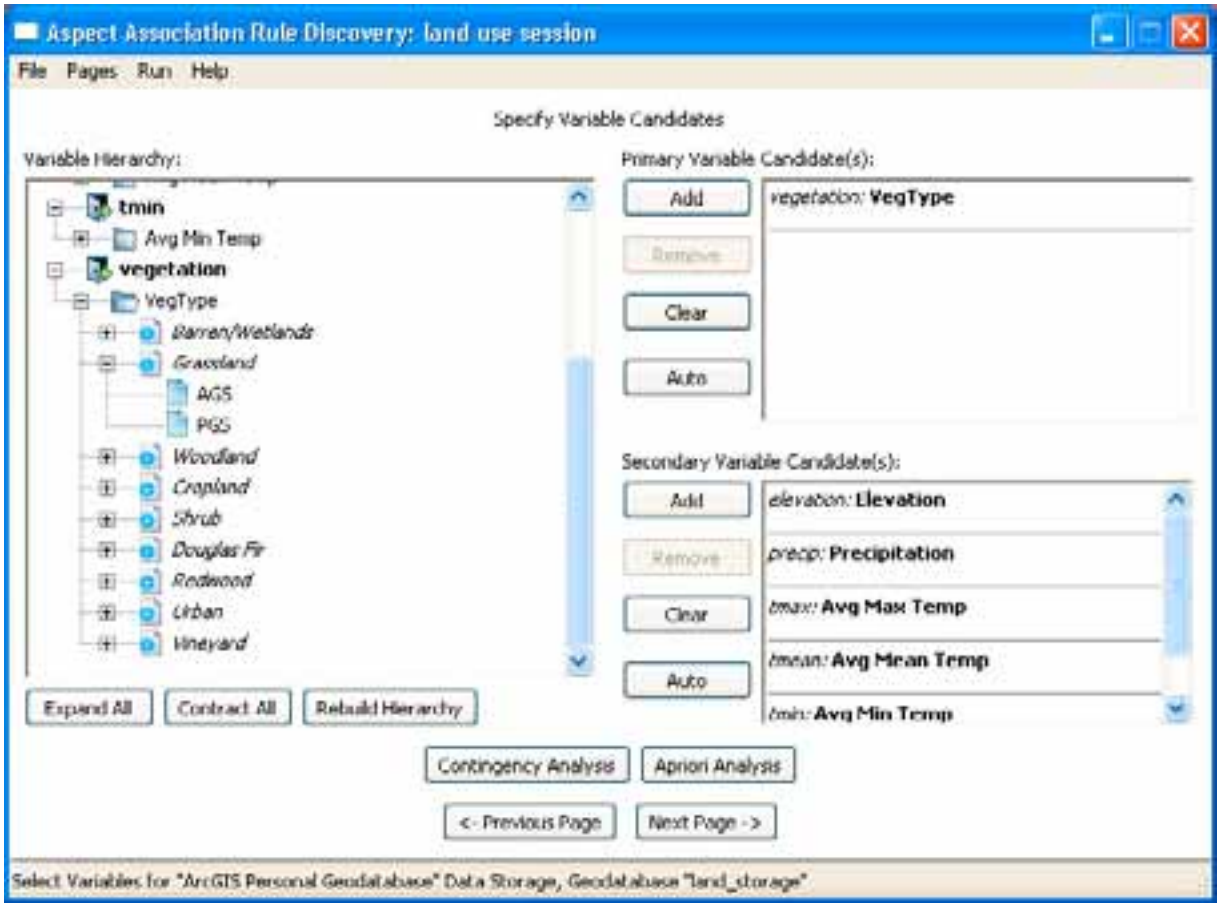


Figure 1.- The "Specify Variable Candidates" window in *Aspect*, showing the hierarchical tree control.

Selections of candidate variables for the association rules are placed in the "primary" or "secondary" variable boxes on the right of Figure 1. After the candidate variables are selected, a contingency table analysis can be performed on the candidate variable combinations. Based on the results of this analysis, attribute values can be grouped together or the candidate variable lists may be modified to better reflect the significant and relevant variables in the associations. Data are passed between the variable selection window and the contingency table window (Figure 2). The contingency table results are formatted using HTML tables. Once the candidate variable list is updated, the Apriori algorithm [4] is called to complete the association rule discovery (Figure 3).

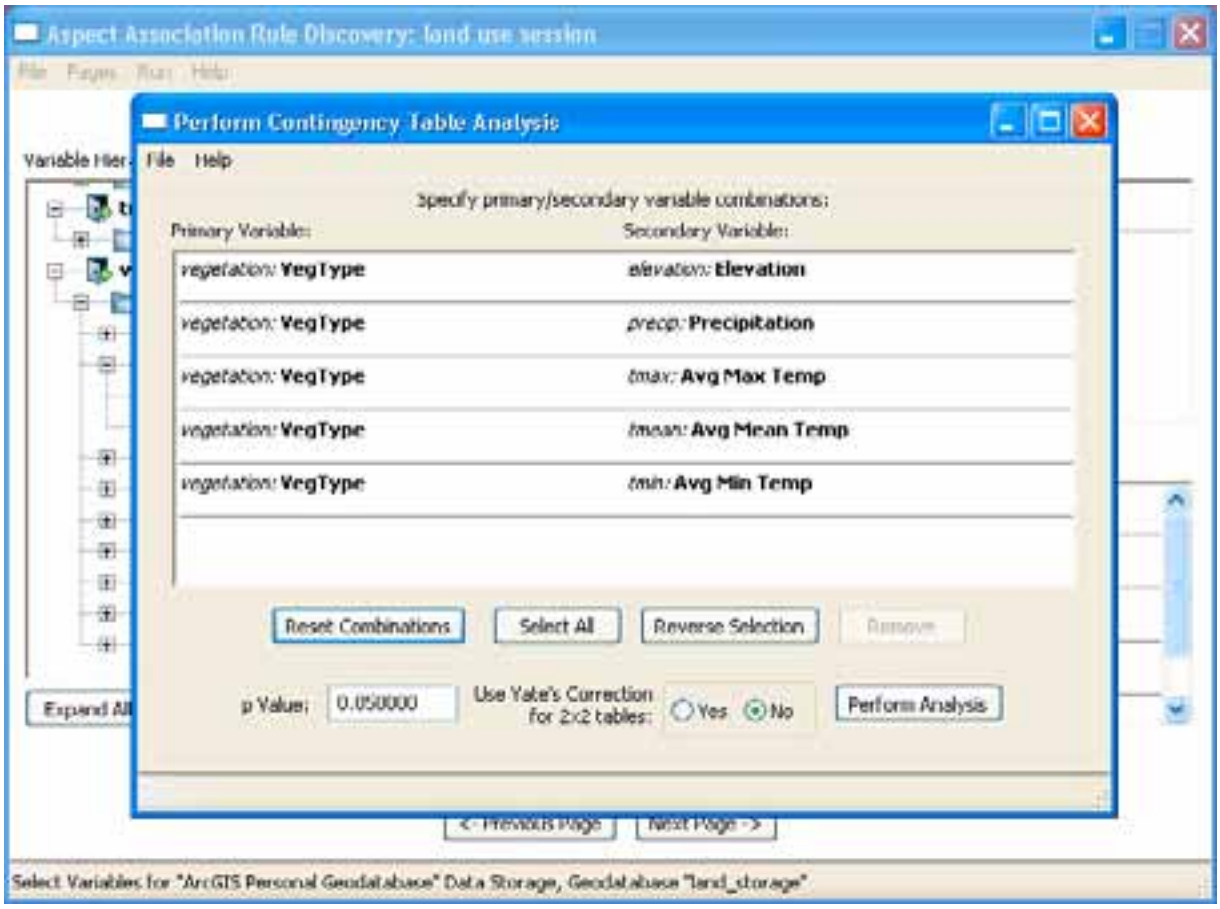
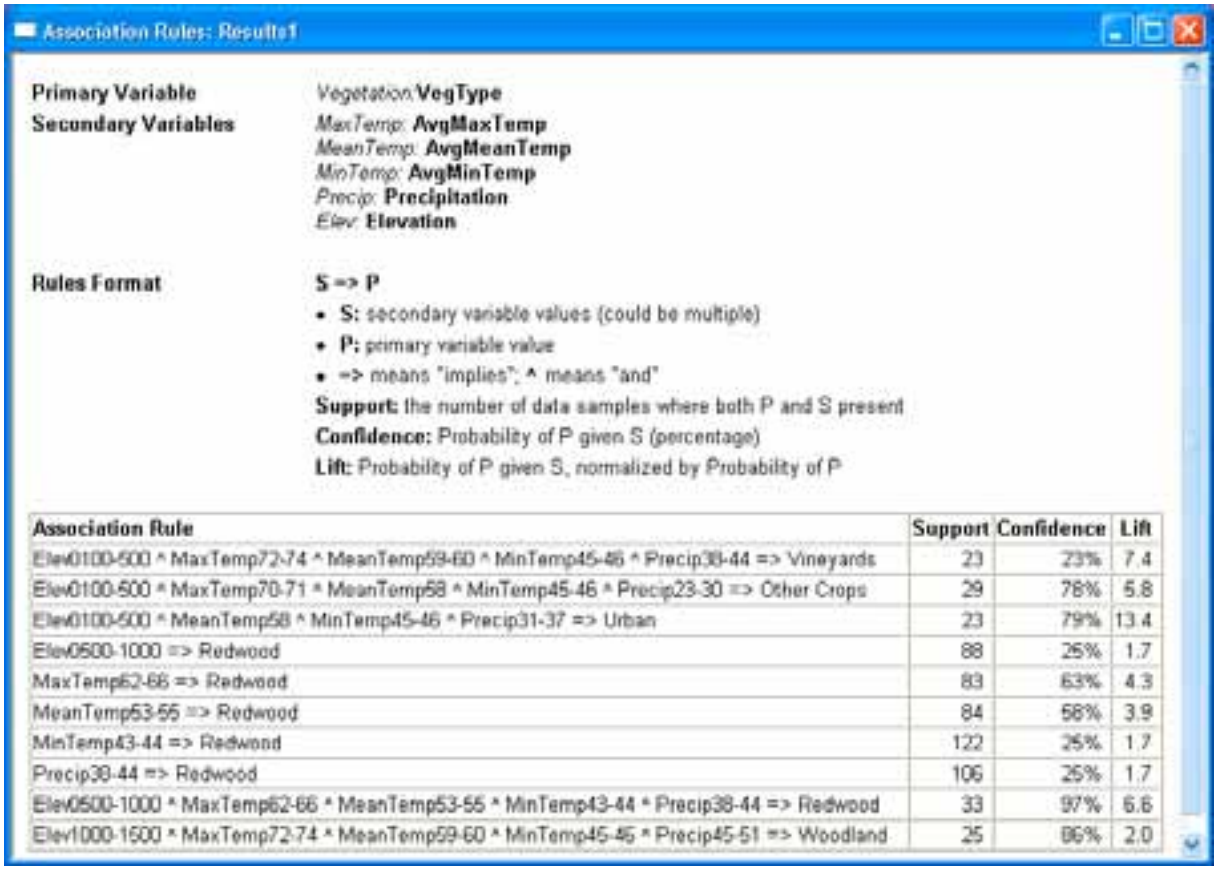


Figure 2.- The "Perform Contingency Table Analysis" window in *Aspect*.

Executable and Installer

To create a standalone application, an executable is created which contains the Python language interpreter, all required libraries, and the source code. By combining everything into one executable, the user is free from dealing with the hassle of libraries and code dependencies. The only additional software that the user is responsible for installing is either ArcGIS or the MySQL database. A free application called cx_Freeze [5] was used to build a Windows version of the executable. Another free application, called Inno Setup [6], was used to build a Windows installer. A wizard guides the user through the installation of *Aspect*.

The installer places shortcuts in the Start Menu to the data mining application, an uninstaller, and any associated files such as data templates and documentation. A help system was written using Microsoft's HTML Help Workshop [7]. The help system has the same look and functionality of other Windows-based help systems, including the desktop help for ArcGIS.



Association Rule	Support	Confidence	Lift
Elev0100-500 ^ MaxTemp72-74 ^ MeanTemp59-60 ^ MinTemp45-46 ^ Precip38-44 => Vineyards	23	23%	7.4
Elev0100-500 ^ MaxTemp70-71 ^ MeanTemp58 ^ MinTemp45-46 ^ Precip23-30 => Other Crops	29	76%	5.8
Elev0100-500 ^ MeanTemp58 ^ MinTemp45-46 ^ Precip31-37 => Urban	23	79%	13.4
Elev0500-1000 => Redwood	88	25%	1.7
MaxTemp62-66 => Redwood	83	63%	4.3
MeanTemp53-55 => Redwood	84	58%	3.9
MinTemp43-44 => Redwood	122	25%	1.7
Precip38-44 => Redwood	106	25%	1.7
Elev0500-1000 ^ MaxTemp62-66 ^ MeanTemp53-55 ^ MinTemp43-44 ^ Precip38-44 => Redwood	33	97%	6.6
Elev1000-1500 ^ MaxTemp72-74 ^ MeanTemp59-60 ^ MinTemp45-46 ^ Precip45-51 => Woodland	25	86%	2.0

Figure 3.- The association rule results window in *Aspect*.

Miscellaneous Tips

The database integration and the GUI are the major components of the Python standalone application, and tips for developing with each are provided in the previous sections. Some other lessons learned during this development effort are described briefly here.

(1) It was noticed that the data mining application would often take a long time to close after the user quit. The reason is because the geoprocessor object writes to a toolbox history file. This file gets larger and larger the more the geoprocessor gets used, and writing to this file takes a long time once it becomes large. As of ArcGIS 9.1, the behavior of writing to the history file cannot be turned off using a script, although it is possible to turn off this behavior manually within ArcGIS. (A fix should be available by version 9.2.) A workaround is used for this problem. When the data mining application starts up, it looks in the registry, finds the history file, and moves it. It then writes its own history to an empty history file. When the

data mining application closes, it moves the original history file back, thus overwriting the history created from the data mining analysis.

(2) ESRI currently does not expose with Python all of the objects that it does with their other APIs. One example is accessing the units of a projected coordinate system. This can be done through the ArcSDE APIs and through the ArcGIS desktop user interface, but not via a Python script.

(3) All of the calls to the ESRI geoprocessing object within *Aspect* are grouped together into a limited number of modules. By containing them in just a few known locations, they will be easier to update in the future should ESRI's Python API change.

(4) Although the data mining application is written to be standalone, it can also be called as a script from within ArcGIS. The application GUI will launch in its own window when the script is run.

Summary

Python can be used to create a spatially enabled standalone program using the ArcGIS geoprocessor. Python has many advantages, since it is open-source, cross-platform, and there is a wealth of available scripts and libraries to extend its functionality with many capabilities.

The data mining application developed here, called *Aspect*, has as major components a relational database management system and a graphical user interface. The database system for the analysis can be either the ArcGIS personal geodatabase or an external database management system such as MySQL. Python provides calls to either database option. The GUI was developed using wxPython, which is based on wxWidgets. This widget library is very flexible and provides all the needed controls for the GUI.

All modules used in *Aspect* are built into a single Windows executable file which can be installed using a wizard. The application can be launched either from the Windows start menu or from inside ArcGIS.

Acknowledgment

This work is sponsored by the U.S. Army Topographic Engineering Center under Contract W9132V-04-C-0025.

References

[1] Rodman, L. C., Jackson, J., Huizar III, R., and Meentemeyer, R. K., "An Association Rule Discovery System for Geographic Data", *Proc. 2006 IEEE International Geoscience and Remote Sensing Symposium*, Denver, CO, July 31-August 4, 2006.

[2] http://www.mysql.com

[3] http://www.wxpython.org

[4] Agrawal, R. and Srikant, R., "Fast Algorithms for Mining Association Rules," *Proc. 1994 Int. Conf. Very Large Data Bases*, Santiago, Chile, pp. 487-499, Sept. 1994.
[5] http://www.python.net/crew/atuining/cx_Freeze/
[6] http://www.jrsoftware.org/isinfo.php
[7] http://www.microsoft.com/downloads/details.aspx?familyid=00535334-c8a6-452f-9aa0-d597d16580cc&displaylang=en

Author Information

Laura C. Rodman
Senior Research Scientist
Nielsen Engineering & Research, Inc.
605 Ellis St., Suite 200
Mountain View, CA 94043
phone: 707-538-8591
fax: 650-968-1410
email: rodman@nearinc.com

John Jackson
Research Engineer
Nielsen Engineering & Research, Inc.
605 Ellis St., Suite 200
Mountain View, CA 94043
phone: 707-538-8591
fax: 650-968-1410
email: jjackson@nearinc.com