Manipulation of Spatial Weights Using Web Services

Sergio J. Rey
GeoDa Center
School of Geographical Sciences & Urban Planning
Arizona State University
Sergio.Rey@asu.edu

Luc Anselin
GeoDa Center
School of Geographical Sciences & Urban Planning
Arizona State University
Luc.Anselin@asu.edu

Myunghwa Hwang
GeoDa Center
School of Geographical Sciences & Urban Planning
Arizona State University
Myunghwa.Hwang@asu.edu

ABSTRACT

In this paper we explore the use of state-of-the-art web technologies to facilitate access to advanced spatial analytical software tools. The specific focus is on the development of web services to disseminate advanced algorithms for spatial weights manipulation. The core functionality is contained in an open source library for spatial analysis called PySAL. Among others, this library provides a variety of methods for creating, transforming, and converting spatial weights. Web services using the Simple Object Access Protocol (SOAP) are created to support machine or developer interfaces to those methods via the Internet. We also develop a web application client for direct access to the weights tools via a browser. The system is illustrated with an example use case in which spatial weights operations are applied to epidemiological data for Ohio counties.

Categories and Subject Descriptors

D.2.12 [**Interoperability**]: Distributed objects;
D.2.13 [**Reusable Software**]: Reusable libraries

General Terms

Algorithms, Design

Keywords

spatial analysis, spatial weights, PySAL, GeoDaWeights, spatial analytical web services

1. INTRODUCTION

Spatial lag operators [2], often referred to as spatial weights matrices, are central in many application areas of spatial analysis. In general terms, a spatial weight $w_{i,j}$ represents a spatial relation between two locations i and j. That spatial relation is often defined in terms of contiguity or distance, although these do not exhaust the possibilities. The collection of all such weights for a system of n locations forms the $n \times n$ spatial weights matrix W. Often these weights are required in the initial phase of a spatial analytical method. For example, in spatial statistics the study of spatial autocorrelation is concerned with various approaches towards estimating the structure and strength of the covariance between a random variable measured at pairs of locations i and j. Given a cross-sectional data set with n observations, there are $(n^2 - n)/2$ pairs of observations and associated covariances, which creates a degrees of freedom problem. A spatial weights matrix is used to reduce the dimensionality of the parameter space. In practice, the information embedded in the weights matrix is not stored as such, but efficient sparse formats such as linked lists or dictionaries are used instead [7].

In the regionalization literature adjacency matrices are key data structures in many algorithms for spatially constrained clustering [13]. Similarly, shortest path finding algorithms make heavy demands on adjacency matrices [1]. Node-node and node-arc adjacency matrices are fundamental to representation of spatial objects and their connectivity in modern geographical information systems [35]. More broadly, adjacency matrices are found throughout the fields of computer graphics and visualization, pattern recognition, and computational geometry [31].

The prominent role of spatial weights matrices across the spectrum of spatial analytical methods requires their implementation in software. This tends to result in considerable duplication, since every method that requires the weights will need to provide code to deal with their construction, manipulation, and transformation. Spatial weights creation and manipulation is still largely absent in todays commercial statistical and econometric software. While specialized packages exist, most are platform-specific or require familiarity with a statistical software library or language (such as R).

In this paper we describe a web services approach to address these issues. We present an approach that builds upon earlier efforts in the delivery of spatial analytical capability through a web interface [6] . Essential analytical algorithms for spatial weights manipulation are encapsulated in the form of a reusable component within the open source library PySAL [29]. A web services interface is designed and implemented to allow the analytical algorithms to be accessed by both end users as well as by other software. The resulting system, including web services and web applications, allows for users with different expertise and across computing platforms to access the analytical tools. In addition, through the web service infrastructure, they can be readily incorporated into other software that adheres to the web service standards.

In the remainder of this paper we first introduce PySAL and its spatial weights core. Next we present the functionality, architecture, and interface of the spatial weights web services system. We illustrate the system with a simple use case where a range of spatial operations is applied to epidemiological data for Ohio counties to yield a spatially lagged variable, or spatial lag (the weighted average of the values observed at neighboring locations). The paper closes with an outline of ongoing and future work.

2. PySAL AND GeoDaWeights

PySAL is an open source library for spatial analysis written in the object-oriented language Python. It is intended to avoid duplication in effort in the development of common core spatial analytical methods, and covers a wide set of areas of spatial analysis as summarized in Figure 1. PySAL is also designed to contribute to the rapidly evolving area of scientific computing with Python [20], in which spatial analysis is still largely absent.

PySAL is designed in a modular fashion with access to the analytical functionality provided via a range of interfaces. Examples include graphical user interfaces (GUIs) built with toolkits such as wxPython [26] in the spatial econometrics toolkit PySpace [5] and TkInter [16] in the exploratory space-time data analysis package STARS [30]. At the same time, users interested in using PySAL through the command line can call PySAL as a standard Python module or, using other shells such as IPython [25] and even from the R [18] interpreter via RPy [24].

A central component in PySAL is a collection of modules to handle spatial weights matrices, referred to as GeoDaWeights. The main functionality of GeoDaWeights includes the creation, characterization, transformation, and conversion of spatial weights as well as spatial lag operations. Due to the importance of spatial weights in many operations and the increasing size of data sets used in applied spatial analysis, there is a need for advanced data structures and efficient geocomputational algorithms to construct spatial weights matrices. GeoDaWeights provides enhanced methods for spatial weights manipulation, in addition to established ones (e.g., [7]). In our system, these functions are delivered as web services. We now briefly describe in turn the three core functions in GeoDaWeights, i.e., the creation, transformation, and conversion of spatial weights.

2.1 Weights creation

Weights creation consists of all the operations necessary to construct a spatial weights object from geographic information on spatial objects. This information is stored either as a boundary file of polygons or as a collection of point coordinates. The operations include reading the geographic information and processing it to derive the topology of the data objects. This then needs to be converted into an efficient data structure.

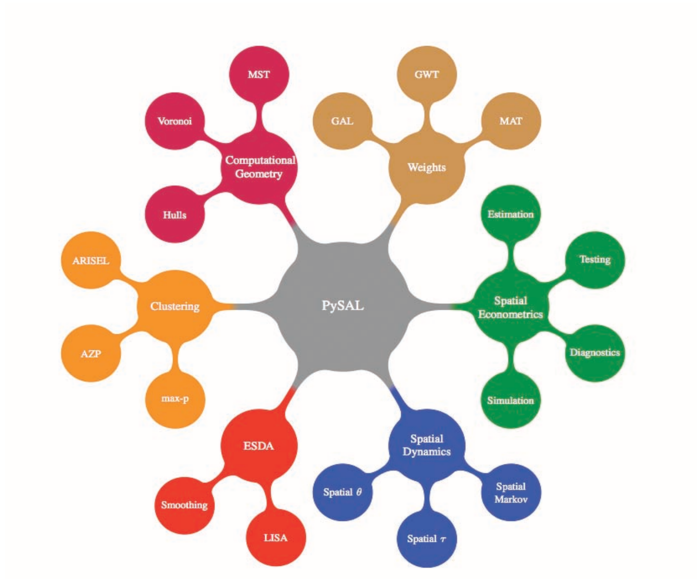


Figure 1: PySAL

Specifically, given a source file with the geographic information, spatial weights creation consists of the following operations:

1. read the geographic information,
2. identify the neighbor relations for each observation following user-specified neighbor criteria,
3. generate an adjacency matrix,
4. apply a weighting method (defined by the user) to the adjacency matrix and generate a spatial weights object,
5. write the spatial weights object to an output file.

In its current implementation, the weights creation component supports neighborhood definitions based on contiguity, distance, and nearest neighbors. These criteria are used to derive the neighbor relations on the fly from standard commercial GIS format files such as ESRI's shapefiles.

An earlier implementation of GeoDaWeights relied on the shapelib library [34] for the reading of shapefiles. However, in the current version we have replaced shapelib with a pure Python implementation of classes to read shapefiles. We have experimented with different algorithms to derive the topology and construct either rook (shared edge) or queen (shared vertices) contiguity matrices. The first method we developed consisted of a binning algorithm that constructs a weights matrix on the fly while reading an input file. It divides the bounding box of the input data into a set of gridded bins to expedite the search of candidate neighbors of each observation. While this algorithm is efficient for small data sets, it is not scalable enough to handle larger data sets (e.g., a boundary file of all U.S. census tracts). In the most recently implemented algorithms this issue is resolved by separating the weights creation from the reading of the file and by adopting advanced data processing techniques such as partial caching of bounding boxes of neighboring polygons and R-tree indexing. All of these algorithms flag island ploygons (i.e., polygons without any contiguity to other polygons). In addition, several options are offered to deal with these island observations or isolates, e.g., following the suggestions in [9].

GeoDaWeights also supports weights construction from shapefiles that contain point data. This includes several methods, using graph based topological criteria such as Gabriel, sphere of influence, and relative neighbor criteria. In addition, k-nearest neighbor algorithms are available for point data.

The point-based methods can also be applied to polygon data, when the latter are represented by points, such as centroid. The weights creation classes in PySAL free the researcher from the often tedious and error-prone tasks involved with constructing spatial weights by hand.

In addition constructing the topological relationships from GIS input files, functionality is also included to directly read common spatial weights formats, such as GAL, GWT, and full matrices. To support combined use of these diverse types of input sources, the latest GeoDaWeights maintains an internal index for observations and their neighbors set so that cross-referencing across observations can be conducted rapidly.

2.2 Weights transformation

As a next step in the processing stack, the transformation component takes the weights generated by the creation component and provides a series of transformations and descriptive statistics of the characteristics of the weights. These include measures of sparseness, distribution of contiguity cardinalities, and various eigenvalue-based metrics of the weights matrix structure.

The transformation functionality includes the construction of higher orders of contiguity, row standardization, conversion of general to binary weights, powering of weights, inverse of weights, algebraic operations on weights, and set based operations (e.g., union, intersection). Users can carry out each of these operations by specifying the operation type in a spatial weights object. Then, the object internally applies the necessary computation and stores the results as its property for further uses.

2.3 Weights conversion

With the proliferation of GIS and spatial analysis software packages, a large variety of formats for spatial weights have appeared. This creates the need to provide functionality to convert between these different formats and allow weights constructed in one package to be applied in analyses carried out in other packages.

Arguably, the most popular format for spatial weights is the GAL format for binary contiguity and the GWT format for general weights. These formats were initially proposed in the SpaceStat software [3, 4] and implemented in the widely distributed GeoDa package [8]. Subsequently, they were adopted by the open source spdep R package as well as the commercial ClusterSeer software.

Other formats include text file formats, such as DAT (MatLab Econometrics Library), TXT (WinBugs), as well as binary formats, including the original SpaceStat matrix format, SWM (ArcGIS 9.3), DBF (ArcGIS 9.3), MAT (MatLab), and WK1 (MatLab Lotus format).

The weights conversion component in GeoDaWeights generates spatial weights objects from these input formats by assigning the information on the spatial adjacency and the value of the weights as properties of the weights objects. The component also includes the functionality to output the weights object into a limited number of commonly used output formats.

3. ARCHITECTURE

The main focus of this paper is on the software architecture needed to move the functionality of GeoDaWeights from the desktop to the internet and to make it accessible by both other programs as well as end users (through a web application interface).

In the following section we explain the general architecture of our implementation of web services and discuss the overall structure of the spatial weights system. This includes a web application which provides a simple browser-based user interface.

3.1 Web services architecture

The concept of a web service refers to web-accessible software systems that provide a variety of service activities through prescribed interfaces and standardized communication protocols [17, 15]. A system architecture based on web services enables dynamic interoperation among multiple software artifacts that run on different platforms and frameworks in distributed networks [10]. A web service, an atom of this architecture, provides a set of operations via web-accessible endpoints and publishes its interface in a machine-processible document. Users then find this service and use its operations by exchanging messages that comply with its published interface. Under this so-called publish–find–bind framework, a web service can be both a client as well as a server. Also, applications can be operationalized by enabling dynamic interactions among services without any centralization of computing resources. This flexibility in the web services architecture, alongside the neutrality across platforms, allows developers to easily share existing software assets and swiftly adapt to constant changes in users' needs [21, 33].

The essential requirement for an effective web services architecture is to ensure interoperability between services. This is generally addressed by developing and adhering to standards for the communication protocol and the interface description. Broadly speaking, present web services can be classified into two distinct groups according to the underlying standards. These categories are referred to as REST (Representational State Transfer) and SOAP (Simple Object Access Protocol) based services. SOAP services are designed to expose the application logic to the Internet. Currently endorsed as a recommendation of the World Wide Web Consortium (W3C), this group of services is built upon a stack of standards such as the Web Service Description Language (WSDL) for interface description [10], SOAP for operation invocation [23], Universal Description, Discovery, and Integration (UDDI) for service search [12], and other WS–* standards for security, reliable messaging, transaction, etc. [14]. In contrast, REST based services focus on the provision of access to data as a resource on the web. In this resource oriented approach every piece of data is assigned a unique Universal Resource Identifier (URI) and manipulated through the universal Hyper Text Transfer Protocol (HTTP) interface [11]. Web service standards developed by the Open Geospatial Consortium (OGC)[1] are considered to weakly follow the REST approach, although it has also been recommended that the OGC services should support SOAP as an optional binding mechanism for message exchange [22, 27].

The spatial weights system is designed on top of the SOAP based web services architecture, for two reasons. First, the focus of the SOAP approach on application logic better suits the goal of the PySAL library, a provision of spatial analysis operations. Second, the standard interface description mechanism of the SOAP architecture facilitates the composition of several distributed services to accomplish new tasks. In other words, new functionality can be created by combining several web services. This modularity matches the principles underlying the PySAL library.

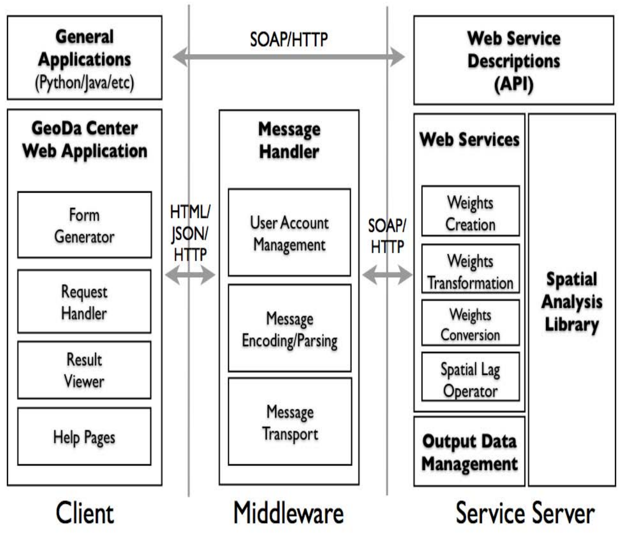


Figure 2: Architecture from a component perspective

3.2 Spatial weights web services architecture

An overview of the architecture of the spatial weights system is given in Figure 2. Three tiers of computing resources are involved, i.e., a server, middleware, and a client interface. The core element of the server tier is the PySAL spatial analytical library that contains the computing functionality to create, transform, and convert spatial weights. A set of web services is wrapped around the functionality of the library to provide the essential application logic of the spatial weights system. Each web service in the system communicates with clients or other web services through SOAP-based messages. A WSDL-based document is also tied to the individual service so that clients can obtain information about its interface such as data types required for input and output messages and protocols for message encoding and transport. This feature is especially useful when client developers have tools that can read WSDL documents and automatically generate code modules for a client application.

In addition to hosting the web service, the server tier also supports data management. This includes a facility to upload user-provided data so that they can be accesses locally by the web services. Also, any output from the services is saved on the server. This can be accessed by users via Universal Resource Locations (URLs). The co-location of services and data storage is intentional in order to circumvent inefficiencies in the transfer of data (such as geographic boundary files and weight files) that occur using Extensible Markup Language (XML) and Javascript Object Notation (JSON) messages. Currently, our system supports only temporary data storage and does not provide any further data management services.

The architecture of the weights system reflects our standpoint that there are generally two types of end users who may interact with the system. In a strict sense, the web services can be seen as an external modular library that can be incorporated into any existing application through the Application Programming Interface (API). In this way, other programs, such as web-aware GIS software, can access the specialized spatial analytical functions included in PySAL through SOAP-based XML messaging. However, this limits use to a relatively small audience of application developers.

In order to widen the dissemination of advanced spatial analytical techniques, we see our analytical web services as an alternative to a desktop application, by providing a means to access this functionality through a browser, via a desktop-like graphical user interface. In contrast to our earlier implementation, through Java-applets that operated on the client desktop [6], we now use a division of labor between the client and the server, with efficient communication between the two.

The middleware tier in the architecture facilitates the necessary communication. It provides operations for message mediation between the front-end client interface and the back-end services. This consists of receiving a simplified message encoded in JSON from the client, and creating and sending a request SOAP message to the server-side services. Conversely, the middleware also receives and parses a response SOAP message from the services and creates and sends a simplified JSON message to the client. In addition, the middleware carries out user authentication by limiting access to the client application to users with the proper privileges. Once authenticated, users can then apply all functions available in the services to their own data sets.

Finally, the client tier contains a GUI that allows users to dynamically interact with each service operation through their browser (see Figure 4). In the GUI, each service is accessed through a menu item that is linked to an *input form* and a *result viewer*. Users generate a request message in

[1] [19] provides a summary of OGC web services with its focus on the Web Processing Service (WPS) standard.

the input form and view the response message in the result viewer. The basic outline of these graphical interfaces was based on the WSDL definition of each service, and then enhanced through further customization. The design is such that the GUI components have no dependency across services so that they can be easily added or removed as the web service interfaces to the PySAL library evolves. In addition to this basic functionality, the GUI includes a set of tutorials detailing the use of the menu items.

The architecture outlined in Figure 2 is enabled by the specific software programs listed in Figure 3. The computational core is provided by the GeoDaWeights component of the PySAL library, which is contained in the server. The functions in this component are wrapped into web services by means of the Python soaplib library[2]. The main role of this library is to serialize Python objects into SOAP-based XML messages (or the reverse) and to automatically generate WSDL documents.

For the middleware tier, we developed several Python Common Gateway Interface (CGI) programs to handle message mediation and user authentication. The *message handler* middleware uses other helper Python libraries, such as simplejson[3] for parsing and encoding JSON messages and Google App Engine webapp module[4] for handling HTTP requests and responses.

For the client tier we utilize the Ext JS Javascript library[5] to develop a desktop-like GUI. The message exchanges between the client and the middleware are supported via familiar Asynchronous Javascript and XML (AJAX) technology, which is embedded in each GUI component.

The GUI and web browser access is just one way to interface with the spatial analytical functionality. Software developers can build their own client applications to access the services directly. For example, the services can be incorporated into a Java application by means of the JAX-WS library[6] or into a Python application by means of the soaplib library.

In the end, this dual approach supports a variety of end users with different levels of technical ability but similar analytical requirements. We consider this in more detail in the next section.

3.3 Web application user interface

The spatial weights services can be accessed directly, in a programmatic manner, or indirectly, through a web application GUI. Direct access is implemented through the exchange of SOAP based XML messages, similar to the way the middleware tier interacts with the services.

However, the easiest access to the analytical functionality is using any standard browser in combination with the client tier.

[2]http://trac.optio.webfactional.com/

[3]http://code.google.com/p/simplejson/

[4]http://code.google.com/p/googleappengine/source/browse/trunk/python/google/appengine/ext/webapp/__init__.py

[5]http://extjs.com/

[6]https://jax-ws.dev.java.net/

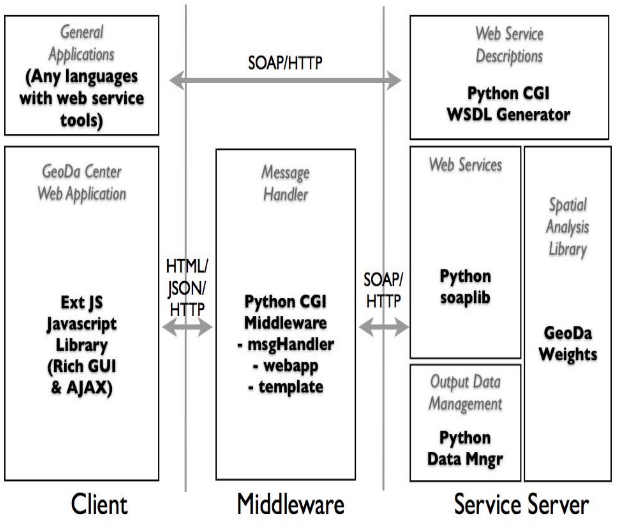
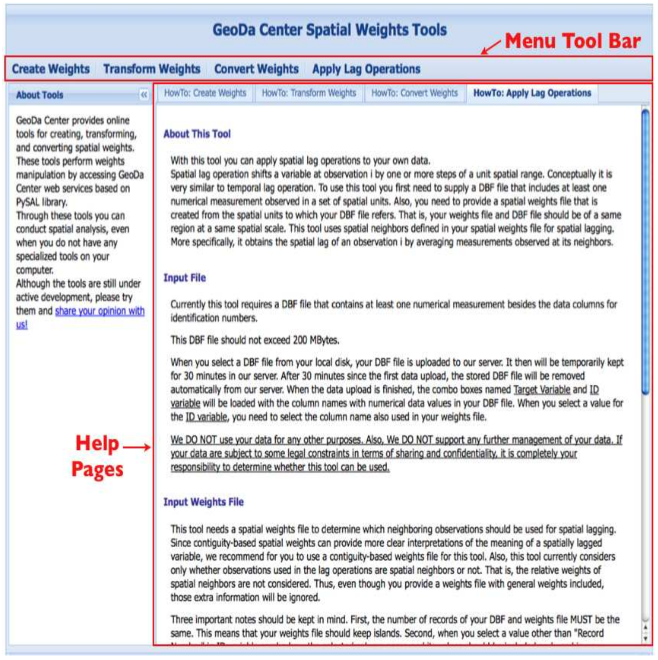


Figure 3: Architecture from a software perspective



GeoDa Center provides online tools for creating, transforming, and converting spatial weights. These tools perform weights manipulation by accessing GeoDa Center web services based on PySAL library.
Through these tools you can conduct spatial analysis, even when you do not have any specialized tools on your computer.
Although the tools are still under active development, please try them and share your opinion with us!

About This Tool

With this tool you can apply spatial lag operations to your own data.
Spatial lag operation shifts a variable at observation i by one or more steps of a unit spatial range. Conceptually it is very similar to temporal lag operation. To use this tool you first need to supply a DBF file that includes at least one numerical measurement observed in a set of spatial units. Also, you need to provide a spatial weights file that is created from the spatial units to which your DBF file refers. That is, your weights file and DBF file should be of a same region at a same spatial scale. This tool uses spatial neighbors defined in your spatial weights file for spatial lagging. More specifically, it obtains the spatial lag of an observation i by averaging measurements observed at its neighbors.

Input File

Currently this tool requires a DBF file that contains at least one numerical measurement besides the data columns for identification numbers.

This DBF file should not exceed 200 MBytes.

When you select a DBF file from your local disk, your DBF file is uploaded to our server. It then will be temporarily kept for 30 minutes in our server. After 30 minutes since the first data upload, the stored DBF file will be removed automatically from our server. When the data upload is finished, the combo boxes named Target Variable and ID variable will be loaded with the column names with numerical data values in your DBF file. When you select a value for the ID variable, you need to select the column name also used in your weights file.

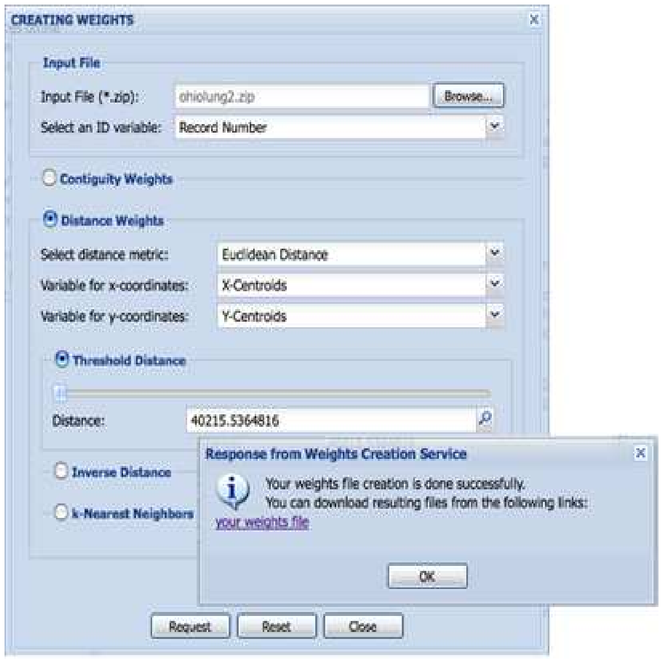
We DO NOT use your data for any other purposes. Also, We DO NOT support any further management of your data. If your data are subject to some legal constraints in terms of sharing and confidentiality, it is completely your responsibility to determine whether this tool can be used.

Input Weights File

This tool needs a spatial weights file to determine which neighboring observations should be used for spatial lagging. Since contiguity-based spatial weights can provide more clear interpretations of the meaning of a spatially lagged variable, we recommend for you to use a contiguity-based weights file for this tool. Also, this tool currently considers only whether observations used in the lag operations are spatial neighbors or not. That is, the relative weights of spatial neighbors are not considered. Thus, even though you provide a weights file with general weights included, those extra information will be ignored.

Three important notes should be kept in mind. First, the number of records of your DBF and weights file MUST be the same. This means that your weights file should keep islands. Second, when you select a value other than "Record

Figure 4: Main GUI for the web client


CREATING WEIGHTS

Input File

Input File (*.zip): ohiolung2.zip [Browse...]

Select an ID variable: Record Number

( ) Contiguity Weights

(●) Distance Weights

Select distance metric: Euclidean Distance

Variable for x-coordinates: X-Centroids

Variable for y-coordinates: Y-Centroids

(●) Threshold Distance

Distance: 40215.5364816

( ) Inverse Distance

( ) k-Nearest Neighbors

[Request] [Reset] [Close]

Response from Weights Creation Service

Your weights file creation is done successfully.
You can download resulting files from the following links:
your weights file

[OK]

Figure 5: Input form (bottom layer) and result viewer (top layer) for the weights creation service

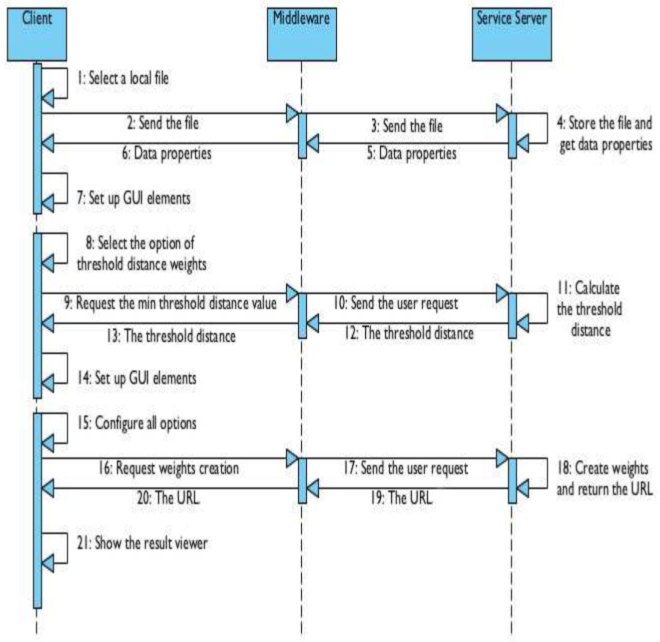

Client
Middleware
Service Server
1: Select a local file
2: Send the file
3: Send the file
4: Store the file and get data properties
5: Data properties
6: Data properties
7: Set up GUI elements
8: Select the option of threshold distance weights
9: Request the min threshold distance value
10: Send the user request
11: Calculate the threshold distance
12: The threshold distance
13: The threshold distance
14: Set up GUI elements
15: Configure all options
16: Request weights creation
17: Send the user request
18: Create weights and return the URL
19: The URL
20: The URL
21: Show the result viewer

Figure 6: The message flow between the web client and the weights creation service

The web client application consists of a base panel and a set of input forms and result viewers. The base panel contains a collection of help pages and a menu tool bar where each item manages a pair of an input form and an associated result viewer to interact with each web service (Figure 4).

When a menu item is clicked, an input form is provided in order to configure input parameters for the corresponding service. In general, an input form is composed of a file uploading element, several selection tools, and a *Request* button. The upload element allows users to move a file from their desktop to the server for processing. The selection tools provide options needed for the specific analytical operations as well as ways to select properties of the output data.

The *Request* button initiates communications with the middleware tier by creating and posting a JSON message that transmits the user selections. Upon receiving the JSON message, the middleware application converts it into a SOAP-based message, which is subsequently dispatched to the target service. Once actual operations are completed in the server, the service sends out a SOAP-based response message to the middleware, which again reformats the message into a JSON object. On the web client, the receipt of the final JSON message invokes a result viewer. This typically provides a list of URLs to access output data on the server.

We illustrate this with an example using the weights creation service. Figure 5 shows the corresponding input form and result viewer. The detailed flow of messages between the different software components is illustrated in Figure 6. In this particular example, the option selected is a so-called threshold distance based spatial weights matrix. In this case, several input parameters for the weights creation depend on the characteristics of the input data. Specifically, in this instance, the service needs to know the type of geometry (polygon or point), the attribute names for the variables and the minimum threshold distance for the distance bands. Since these parameters depend on the input data, they are not known until after the files have been uploaded to the server. The selection tools in the input form dynamically set some of the option values by communicating with the web service that stores user data on the server and extracts the basic features of the data. As a result, the input form carries out multiple message exchanges with other services before it conducts the main operation related to the weights creation itself.

4. ILLUSTRATION

We provide a brief illustration of the functionality of the spatial weights web services system with a calculation of so-called spatially lagged variables (weighted averages of the neighboring observations) using a well know epidemiological data set on lung cancer in Ohio counties. The data set contains geographic boundary information, population, and the number of lung cancer patients for Ohio counties in 1968, 1978, and 1988.[7]

We show two examples of the use of the services. First, we use the weights creation service and the spatial lag service sequentially. In a programmatic environment, this could be implemented in a script that chains the two services. We compute a spatially lagged variable for the number of male lung cancer patients in 1988 ("LM88"). This requires a spatial weights matrix as input. In our example, we first use simple contiguity weights.

[7]This data set can be downloaded from the homepage of GeoDa Center (http://geodacenter.asu.edu/sdata).

After selection of the menu item “Create Weights,” the weights creation service interface appears, as illustrated in the left panel of Figure 7. As input data, the service accepts a ZIP file that contains SHP, SHX, and DBF files with the information on the polygon boundaries and the attributes in the standard ESRI shape file format. The ZIP file of the sample Ohio data is selected by clicking “Browse” button. Immediately, the selected data set is uploaded onto the server and the drop down list for the ID variable is filled with the names of its numerical attributes. In this illustration “FIPSNO” is used for the ID variable of the output weights matrix file. Next, we select the radio button for the “Contiguity Weights” option and use the default values for the other options, i.e., “Rook” and “1” for the type and order of contiguity. After pressing the “Request” button, we obtain the URL to the resulting GAL file so that the file can be saved to a local disk.

To construct the actual spatially lagged variable, we select the menu item labelled “Apply Lag Operations.” As shown in the the right hand panel of Figure 7, four items need to be specified: the input data file, the target variable, the ID variable and the input spatial weights file. In our example, the DBF file in the Ohio data set is the input data file, and “LM88” and “FIPSNO” are, respectively, the target variable and ID. The just created GAL spatial weights file is selected as the weights input. Proceeding with *Request* yields the URL to a DBF file that contains a new lag variable.

In a second example, we compute the spatially lagged variable in the same manner, but use input created by the weights transformation and weights conversion services. In order to accomplish this, we will not create the spatial weights from a polygon boundary file, but instead we will use the existing GAL file and transform it into a different type of weights. Specifically, we use the “Transform Weights” menu item to transform the first order contiguity matrix just created into a second order (neighbors of neighbors) matrix.

As illustrated in the left panel of Figure 8, the just created GAL file is specified as input and several options are set. This includes “Higher Order Contiguity” for the transformation option, “2” as the order of contiguity and “GWT” as the output file format. After sending the *Request*, a URL is provided to the computed GWT file, which can then be stored locally. The spatially lagged variable is computed using the “Apply Lag Operations” tool in the same way as for the previous example.

Finally, to illustrate the weights conversion functionality, we show in Figure 8 how the first order contiguity weights in GAL format are turned into a DBF format. The right hand panel in the figure shows the input file and “ArcGIS DBF” as the target format. The resulting DBF file can then again be used as the input weights file for the computation of a spatially lagged variable.

5. CONCLUSION

The web services spatial weights system described here is an initial step in providing a network distributed interface to the PySAL library. Designing it around the weights core of PySAL reflects the central place of spatial weights in much of spatial analysis. In subsequent research we intend to continue to refine the web-based interface to the weights core and eventually begin to extend the same architecture to provide access to other components of the PySAL library from Figure 1. This line of our efforts, in the long run, would also grow into the developing of a platform where our web service components can be dynamically integrated into new computational resources.

While we feel the computational components of PySAL lend themselves very nicely to being integrated in the web services system, we see a major challenge in extending the system to support the kind of interactive and dynamic graphics available in desktop spatial analysis packages such as GeoDa [8] and STARS [30]. Parallel to the work reported here is ongoing research on efficient web-based geovisualization methods [32] that we plan to explore in addressing these challenges.

Finally, while much of the work described here focuses on the analytical dimensions of the interface, we are keenly aware of the role that open source can play in facilitating learning and advances in scientific communities. Spatial analysis as a web service poses particular challenges in this regard as it is the underlying functionality of PySAL that is made available to the end users and not the source code and implementation [28]. We are currently exploring approaches towards providing source code and documentation browsing, alongside the analytical functionality, as components of the next generation of web services.

6. ACKNOWLEDGEMENTS

This research was supported in part by a grant from the National Cancer Institute (1R01CA126858) for the projects entitled “Geospatial factors and impacts: measurement and use.” The content is solely the responsibility of the authors and does not necessarily represent the official views of the National Cancer Institute or the National Institutes of Health.

7. REFERENCES

[1] R. Ahuja, K. Mehlhorn, J. Orlin, and R. Tarjan. Faster algorithms for the shortest path problem. *Journal of the ACM (JACM)*, 37(2):213–223, 1990.

[2] L. Anselin. *Spatial Econometrics: Methods and Models*. Kluwer Academic, Dordrecht, The Netherlands, 1988.

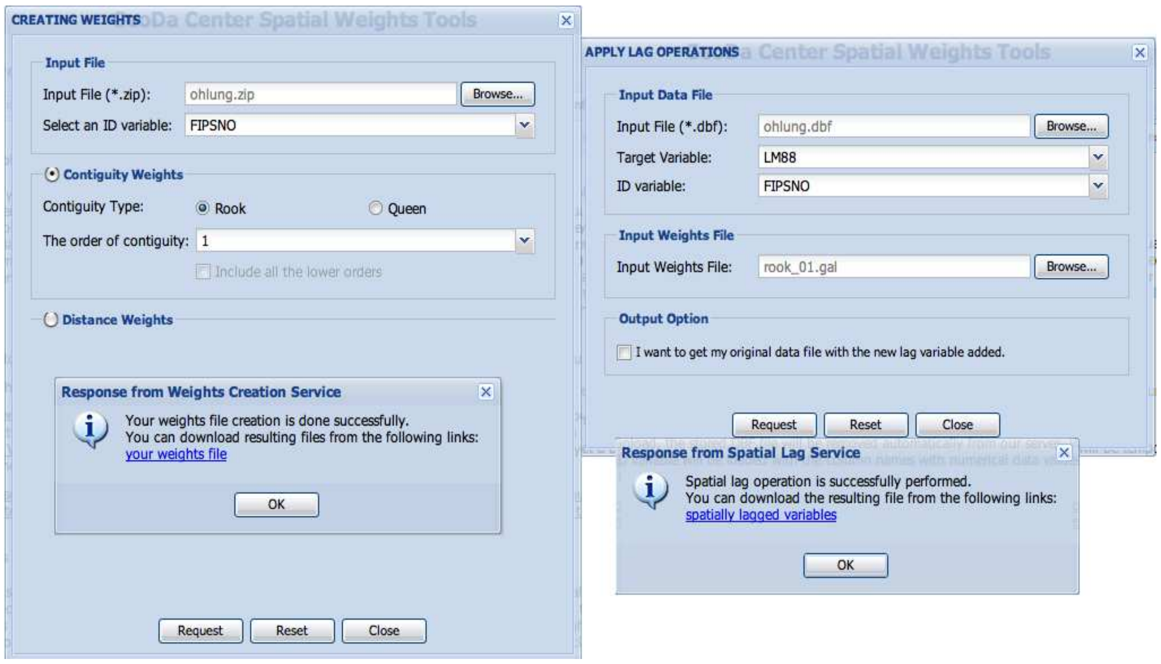
[3] L. Anselin. *SpaceStat, a Software Program for Analysis of Spatial Data*. National Center for Geographic Information and Analysis (NCGIA), University of California, Santa Barbara, CA, 1992.

[4] L. Anselin. Computing environments for spatial data analysis. *Journal of Geographical Systems*, 2(3):201–220, 2000.

[5] L. Anselin and J. L. Gallo. Panel data spatial econometrics with pyspace. Technical report, Department of Agricultural and Consumer Economics, University of Illinois, Urbana-Champaign, IL, 2004.

[6] L. Anselin, Y.-W. Kim, and I. Syabri. Web-based analytical tools for the exploration of spatial data. *Journal of Geographical Systems*, 6:197–218, 2004.

[7] L. Anselin and O. Smirnov. Efficient algorithms for constructing proper higher order spatial lag operators.



CREATING WEIGHTS

Input File

Input File (*.zip): ohlung.zip [Browse...]

Select an ID variable: FIPSNO

(•) Contiguity Weights

Contiguity Type: (•) Rook ( ) Queen

The order of contiguity: 1

[ ] Include all the lower orders

( ) Distance Weights

Response from Weights Creation Service

Your weights file creation is done successfully.
You can download resulting files from the following links:
your weights file

[OK]

[Request] [Reset] [Close]

APPLY LAG OPERATIONS

Input Data File

Input File (*.dbf): ohlung.dbf [Browse...]

Target Variable: LM88

ID variable: FIPSNO

Input Weights File

Input Weights File: rook_01.gal [Browse...]

Output Option

[ ] I want to get my original data file with the new lag variable added.

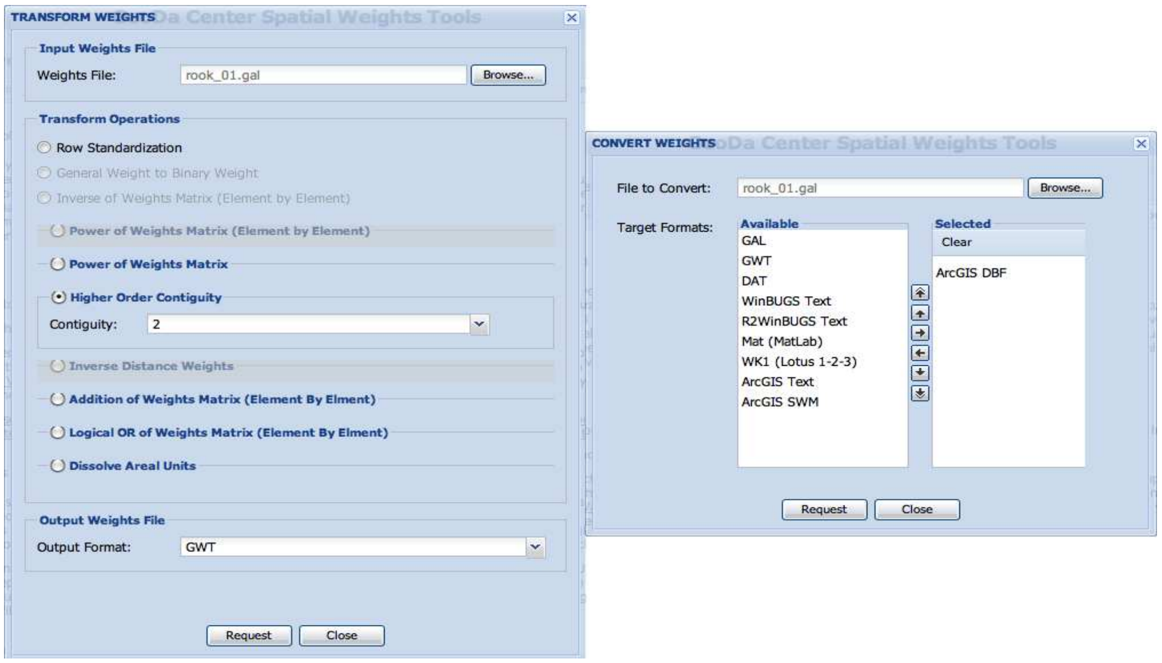
[Request] [Reset] [Close]

Response from Spatial Lag Service

Spatial lag operation is successfully performed.
You can download the resulting file from the following links:
spatially lagged variables

[OK]

Figure 7: A spatial lag operation using a contiguity weights matrix



TRANSFORM WEIGHTS

Input Weights File

Weights File: rook_01.gal [Browse...]

Transform Operations

( ) Row Standardization

( ) General Weight to Binary Weight

( ) Inverse of Weights Matrix (Element by Element)

( ) Power of Weights Matrix (Element by Element)

( ) Power of Weights Matrix

(•) Higher Order Contiguity

Contiguity: 2

( ) Inverse Distance Weights

( ) Addition of Weights Matrix (Element By Elment)

( ) Logical OR of Weights Matrix (Element By Elment)

( ) Dissolve Areal Units

Output Weights File

Output Format: GWT

[Request] [Close]

CONVERT WEIGHTS

File to Convert: rook_01.gal [Browse...]

Target Formats:

Available: GAL, GWT, DAT, WinBUGS Text, R2WinBUGS Text, Mat (MatLab), WK1 (Lotus 1-2-3), ArcGIS Text, ArcGIS SWM

Selected: Clear; ArcGIS DBF

[Request] [Close]

Figure 8: GUIs for the weights transformation and conversion services

Journal of Regional Science, 36:67–89, 1996.

[8] L. Anselin, I. Syabri, and Y. Kho. Geoda: An introduction to spatial data analysis. *Geographical Analysis*, 38:5–22, 2006.

[9] R. Bivand and B. Portnov. Exploring spatial data analysis techniques using R: The case of observations with no neighbors. In L. Anselin, R. J. G. M. Florax, and S. J. Rey, editors, *Advances in Spatial Econometrics: Methodology, Tools and Applications*, pages 121–142. Springer, 2004.

[10] D. Booth and C. K. Liu. Web service description language (WSDL) version 2.0 part 0: Primer. Technical report, 2007.

[11] D. Chappell. SOAP vs. REST: Complements or competitors? http://www.esri.com/events/devsummit/ pdfs/keynote_chappell.pdf, 2009.

[12] L. Clement, A. Hately, C. von Riegen, and T. Rogers. UDDI version 3.0.2. Technical report, 2004.

[13] J. Duque, R. Ramos, and J. Surinach. Supervised regionalization methods: A survey. *International Regional Science Review*, 30(3):195, 2007.

[14] T. Erl. *Service-oriented Architecture (SOA): Concepts, Technology, and Design*. Prentice Hall, 2004.

[15] D. Fensel, H. Lausen, A. Polleres, J. Brujin, M. Stollberg, D. Roman, and J. Domingue. *Enabling Semantic Web Services: The Web Service Modeling Ontology*. Springer, 2007.

[16] J. Grayson. *Python and Tkinter Programming*. Manning, Greenwich, CT, 2000.

[17] H. Hass and A. Brown. Web services glossary. 2004.

[18] R. Ihaka and R. Gentlman. R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics*, 5:299–314, 1996.

[19] C. Kiehle, C. Heier, and K. Greve. Requirements for next generation spatial data infrastructures - standardized web based geoprocessing and web service orchestration. *Transactions in GIS*, 11(6):819–834, 2007.

[20] H. Lantangen. *Python Scripting for Computational Science*. Springer, 2006.

[21] P. A. Longley, M. F. Goodchild, D. J. Maguire, and D. W. Rhind. *Geographic Information Systems and Science*. Wiley, 2005.

[22] R. Lucchi, M. Millot, and C. Elfers. Resource oriented architecture and REST: Assessment of impact and advantages on INSPIRE. Technical report, EUR 23397 EN, 2008.

[23] N. Mitra and Y. Lafon. SOAP version 1.2 part 0: Primer. Technical report, 2007.

[24] W. Moreira and G. Warnes. Rpy (R from Python). Technical report, 2006.

[25] F. Perez and B. Granger. Ipython: A system for interactive scientific computing. *Computing in Science and Engineering*, 9(3):21–29, 2007.

[26] N. Rappin and R. Dunn. *wxPython in action*. Manning, Greenwich, CT, 2006.

[27] C. Reed. Specification best practices. Technical report, Open Geospatial Consortium, 2006.

[28] S. J. Rey. Show me the code: Open source and spatial data analysis. *Journal of Geographical Systems*, 11(2):191–207, 2009.

[29] S. J. Rey and L. Anselin. PySAL: A python library for spatial analytical methods. *The Review of Regional Studies*, 37:5–27, 2007.

[30] S. J. Rey and M. V. Janikas. STARS: Space-time analysis of regional systems. *Geographical Analysis*, 38:67–86, 2006.

[31] H. Samet. *Foundations of multidimensional and metric data structures*. Morgan Kaufmann, San Francisco, 2006.

[32] C. Schmidt and B. Dev. A scalable tile map service for distributing dynamic choropleth maps. Technical report, GeoDa Center for Geospatial Analysis and Computation, Arizona State University, 2008. Working Paper 2008-13.

[33] M. Tsou and B. P. Buttenfield. A dynamic architecture for distributing geographic information services. *Transactions in GIS*, 6(4), 2002.

[34] F. Warmerdam. Shapefile c library v1.2. http://shapelib.maptools.org/, 2006.

[35] M. Worboys and M. Duckham. *GIS, A computing perspective*. CRC Press, Boca Raton, 2004.