Plugins and Scripts Developed to Conduct Space Syntax Analysis in FOSS4GIS:

OpenJUMP, gvSIG, OrbisGIS, Quantum GIS, OpenEV, Thuban, MapWindow GIS, SAGA, and R Project
Plugins and Scripts Developed to Conduct Space Syntax Analysis in FOSS4GIS:

OpenJUMP, gvSIG, OrbisGIS, Quantum GIS, OpenEV, Thuban, MapWindow GIS, SAGA, and R Project

BURAK BEYHAN
Assistant Professor in the Department of City and Regional Planning at Mersin University

Mersin University Publication
Publication No. 34
Preface

This book is the outcome of a project initiated by me three years ago in order to create plugins for Space Syntax Analysis (SSA) by using Free and Open Source Software (FOSS) for Geographic Information Systems (GIS). My departure point for the project was to uncover a block box that can be used in the analysis of the underlying reasons between social and spatial configurations embedded in a region, city or more particularly district. Until recently majority of the software programs available for the analysis of spatial configurations in terms of SSA were not open source, albeit freely available for academic purposes.

At the beginning of the project a wide range of FOSS4GIS programs were under consideration to develop a SSA Plugin. Nevertheless, some of them were omitted from the project due to the unresolved issues that the author experienced in the respective programs. The author himself has explored many things during his engagement in different scripting languages through which the plugins have been developed for different FOSS4GIS. Thus, the book reflects the experiences of the author along the process throughout which SSA Plugins have been created for some FOSS4GIS.

In this respect, the book can serve for different purposes. On the one hand, it provides a background for those willing to develop (graph theoretic) plugins by using FOSS4GIS (particularly the ones used in this book). On the other hand, it develops a concern for the analysis of spatial configurations via SSA by using the plugin designed and created for this purpose in this book. I hope that this introductory book will be useful for those willing to use SSA Plugin in terms of its theoretical background and the tools developed for the exchange of information between SSA and Social Network Analysis (SNA). It is also hoped that the users will develop a simple understanding of how to develop similar kinds of plugins by using FOSS4GIS.

Burak Beyhan
Mersin, 2012
Acknowledgments

I would like to thank Mersin University for supporting the scientific research project (BAP-MF ŞBP (BB) 2010-4 A) contributing to the production of this book, also published by Mersin University using the budget allocated for this purpose in the project.

I would also like to thank FOSS and FOSS4GIS communities for supporting the free spread of ideas and knowledge through the software repositories, and discussion forums and platforms that have been used in developing SSA plugins and scripts for FOSS4GIS.
Preface ......................................................................................i
Acknowledgments ...................................................................ii
Content................................................................................... iii
Tables.......................................................................................v
Figures ....................................................................................vi
Scripts .....................................................................................ix
Acronyms................................................................................xi
1. Introduction..........................................................................1
2. Why FOSS4GIS for SSA Plugin?........................................6
3. Basic Definitions in Space Syntax Analysis and Need for Interaction between SSA and SNA .......................12
   3.1. What is Adjusted Graph?............................................12
   3.2. Parameters Used in SSA in Connection with SNA ....14
   3.3. Options for the Construction of Line Networks.........17
   3.4. Necessity of Interaction between SSA Plugin and SNA Software Programs .....................................19
4. Conceptualization of SSA in GIS and Structure of SSA Scripts and Plugins..............................................21
   4.1. Construction of Adjusted Graph.................................24
      4.1.1. Using Built-in Functions to Check Intersection.....25
      4.1.2. Calculation of Bounding Boxes...........................26
      4.1.3. Checking Overlapping between Bounding Boxes..28
      4.1.4. Checking Intersection between Line Segments .....29
   4.2. Calculation of Geodesic Distance ..............................32
   4.3. Calculation of Parameters and Writing Results to Attribute Table .....................................................35
   4.4. Calculation of Intelligibility Value.............................37
   4.5. Exporting Adjusted Graph and Geodesic Distance into a SNA File Format ............................................39
Tables

Table 1. OGR Vector Formats..........................................................7
Table 2. Scripting opportunities and license types in FOSS4GIS..........................................................8
Table 3. Advantages and disadvantages of FOSS versus proprietary software...........................................11
Table 4. SNA file formats chosen for SSA Plugin..................20
Table 5. Different options that may be followed in FOSS4GIS for conducting SSA..................................21
Table 6. Compatibility of SSA Plugin with FOSS4GIS...........46
Table 7. SSA Plugin files or folder to copy and the location of plugin directory for various FOSS4GIS (Windows)......98
Table 8. SSA Plugin files or folder to copy and the location of plugin directory for various FOSS4GIS (Ubuntu)........99
Table 9. Menu items and toolbar through which SSA Plugin can be activated and run in various FOSS4GIS........100
Table 10. Links for FOSS4GIS on which SSA Plugin is operational..........................................................105
Table 11. Menu and mouse button path to produce thematic maps in various FOSS4GIS.............................106
Table 12. Types of software licenses for various network analysis software programs..................................112
Table 13. Loading adjusted graph saved in “.net” file format into various network analysis software programs........113
Table 14. Loading adjusted graph saved in “.dl” file format into various network analysis software programs.........114
Table 15. Importing tabular data into FOSS4GIS..........................120
Figures

Figure 1. Four land areas and seven bridges in the 17th century Königsberg..................................................................1

Figure 2. Schematic (left diagram) and graph theoretical (right) representations of Königsberg’s Seven Bridges Problem.................................................................2

Figure 3. A simple historical sketch for the socio-spatial methods of analysis based on graph theory. .........................2

Figure 4. Spatial configuration of street segments in Siteler. .....12

Figure 5. Street segments at the north-west of Siteler...........13

Figure 6. Adjusted graph created for the street segments located at the north-west of Siteler. .................................13

Figure 7. Sample adjusted graph. ...........................................20

Figure 8. Algorithm employed for the creation of SSA Plugin.................................................................23

Figure 9. Graphical User Interface of SSA Plugin in Jython for OpenJUMP. .................................................................43

Figure 10. Importing an existing project into Eclipse...........49

Figure 11. Eclipse IDE after opening the existing workspace....50

Figure 12. Selecting “Properties” in order to add external JARs into a project in Eclipse.................................52

Figure 13. Adding external JARs into a project in Eclipse.......53

Figure 14. Creating a package and Class file under a project in Eclipse.................................................................55

Figure 15. Creating a new Java Class for a project in Eclipse.....56

Figure 16. BeanShell Console in OpenJUMP.........................57

Figure 17. GUI of BeanShellEditorPlugin in OpenJUMP.......58

Figure 18. Construction of adjusted graph and calculation of geodesics by using BeanShell Console in OpenJUMP....59

Figure 19. Jython Console in OpenJUMP..............................60
Figure 20. Construction of adjusted graph and calculation of geodesics by using Jython Console in OpenJUMP ............. 61
Figure 21. Jython Console in gvSIG........................................... 62
Figure 22. GUI alternatives in gvSIG ......................................... 64
Figure 23. Auto-complete feature of Jython Console in gvSIG .................................................................................... 65
Figure 24. Construction of adjusted graph by using Jython Console in gvSIG........................................................... 65
Figure 25. Calculation of geodesics by using Jython Console in gvSIG............................................................. 66
Figure 26. ThinG for creating xml GUIs. ................................. 67
Figure 27. BeanShell Console in OrbisGIS................................ 70
Figure 28. Construction of adjusted graph and calculation of geodesics by using BeanShell Console in OrbisGIS........ 71
Figure 29. Creating GUI of SSA Plugin by using BeanShell Console in OrbisGIS.................................................. 72
Figure 30. GUI of QGIS Plugin Builder................................. 73
Figure 31. QT Creator for creating Python GUIs.................... 74
Figure 32. Python Console in Quantum GIS.......................... 75
Figure 33. Construction of adjusted graph by using Python Console in Quantum GIS. ............................................ 76
Figure 34. Calculation of geodesics by using Python Console in Quantum GIS........................................................ 76
Figure 35. Python Console in OpenEV .................................. 79
Figure 36. Iterating over geographic features in a vector layer by using Python Console in OpenEV ....................... 80
Figure 37. Scripting tool in MapWindow GIS ........................... 81
Figure 38. VB.Net script produced for iteration of features in MapWindow GIS in order to check intersection between BBs ........................................................................................................ 83
Figure 39. Microsoft Visual C++ for developing and compiling plugins for SAGA GIS........................................... 85
Figure 40. R editor to run and produce R scripts...................... 94
Plugins and Scripts Developed to Conduct SSA in FOSS4GIS

Figure 41. R script produced for iteration of features in a vector layer in order to check the intersection between bounding boxes. .................................................................95

Figure 42. Parameters calculated by SSA Plugin and its GUI. 102

Figure 43. A thematic map produced in OpenJUMP for Global Integration values. .................................................107

Figure 44. A thematic map produced in Quantum GIS for Global Integration values. ..................................................107

Figure 45. A thematic map produced in Thuban for Global Integration values. ........................................................108

Figure 46. A thematic map produced in OpenEV for Global Integration values. ........................................................108

Figure 47. A thematic map produced in gvSIG for Global Integration values. ..........................................................109

Figure 48. A thematic map produced in SAGA GIS for Global Integration values. .....................................................109

Figure 49. A thematic map produced in MapWindow GIS for Global Integration values. ........................................110

Figure 50. A thematic map produced in OrbisGIS for Global Integration values. .....................................................110

Figure 51. A thematic map produced in R Project for Global Integration values. .....................................................111
Scripts

Script 1. Construction of adjusted graph in OpenJUMP BeanShell. ................................................................. 26
Script 2. Calculation of bounding boxes in OpenEV. .......... 27
Script 3. Checking overlapping between bounding boxes in OpenEV. ............................................................... 28
Script 4. Checking overlapping between BBs in Thuban. .... 28
Script 5. Checking intersection between line features in Thuban. ................................................................. 30
Script 6. Calculation of geodesics in Python. ......................... 33
Script 7. Calculation of geodesics in Jython. ........................ 34
Script 8. Calculation of parameters and writing them to attribute table in OpenJUMP BeanShell. .................... 35
Script 9. Jython script for Pearson correlation function. ....... 38
Script 10. BeanShell script for Pearson correlation function. .... 39
Script 11. Jython script created for defining header information and labeling of column headings in OpenJUMP. ................................................................................. 40
Script 12. Jython script for GUI in OpenJUMP. .................... 44
Script 13. Summarized content of MekanDizimPlugIn.java. .... 54
Script 14. Jython script for GUI in gvSIG. ............................. 68
Script 15. Python script created for iterating over the features covered by a vector layer in Thuban. .................... 78
Script 16. Python script created for defining the menu entry and the handle for SSA Plugin in OpenEV.............. 80
Script 17. Content of MLB_Interface.cpp for SSA Plugin. ...... 86
Script 18. Constructor part of space_syntax.cpp for SSA Plugin. ....................................................................... 88
Script 19. Visual C++ code produced in order to define variables. ................................................................. 90
Plugins and Scripts Developed to Conduct SSA in FOSS4GIS

Script 20. Visual C++ code produced for iteration of features in SAGA GIS in order to check intersection between bounding boxes. .................................................................92

Script 21. Visual C++ code produced to create new fields in the attribute table for space syntax parameters..................93
Acronyms

API: Application Programming Interface
ASCII: American Standard Code for Information Interchange
AWT: Abstract Window Toolkit
BB: Bounding Box
BSU: Basic Spatial Unit
CAD: Computer Assisted Drawing
CDDL: Common Development and Distribution License
CSV: Comma Separated Value
DCM: Digital Cartographic Model
DGN: MicroStation Design File
DODS: Distributed Oceanographic Data System
DWG: AutoCAD Drawing File
DXF AutoCAD Data Exchange File
E00: ArcInfo Export File
EDIGEO: Échange de Données Informatisées dans le Domaine de l'Information Géographique
Epi Info: Public domain statistical software for epidemiology
Epi Info REC: Epi Info record file
ESRI: Environmental Systems Research Institute
FAA: Federal Aviation Administration
FME: Feature Manipulation Engine
FOSS: Free and Open Source Software
FOSS4GIS: Free and Open Source Software for GIS
GDAL: Geospatial Data Abstraction Library
GeoRSS: Geographically Encoded Objects for RSS feeds
GEOS: Geometry Engine – Open Source
GIMP: GNU Image Manipulation Program
GIS: Geographic Information System
GML: Geography Markup Language
GMT: Generic Mapping Tools
Plugins and Scripts Developed to Conduct SSA in FOSS4GIS

GNU: GNU is not Unix
GPL: General Public License
GPS: Global Positioning System
GPX: GPS Exchange Format
GRASS: Geographic Resources Analysis Support System
GTM: GPSTrackMaker
GUI: Graphical User Interface
gvSIG: Generalitat Valenciana Sistemes d’Indormació Geogràfica
IDE: Integrated Development Environment
Ingres: INteractive Graphics REtrieval System
INTERLIS: A Data Exchange Mechanism for Land-Information Systems
JSON: JavaScript Object Notation
JTS: Java Topology Suite
JUMP: Java Unified Mapping Platform
JYTHON: Java-Based Python
KML: Keyhole Markup Language
LGPL: Lesser General Public License
MDB: Microsoft Access Database
MPL: Mozilla Public License
NTF: Neutral Transfer Format (formerly National Transfer Format) used by the Ordnance Survey in Britain.
NWB: Network Workbench
OGC: Open Geospatial Consortium
OGDI: Open Geographic Datastore Interface
OGR: OpenGIS Simple Features Reference Implementation
OPeNDAP: Open-source Project for a Network Data Access Protocol
ORA: Organizational Risk Analyzer
PDS: Photogrammetric Data Services
postgre: “post-Ingres” Database
Proj4: Cartographic Projections Library
QGIS: Quantum GIS
Acronyms

RSS: Really Simple Syndication or Rich Site Summary
SAGA: System for Automated Geoscientific Analyses
SDE: Spatial Database Engine
SDK: Software Development Kit
SDTS: Spatial Data Transfer Standard
SEG-Y: Society of Exploration Geophysicists format Y
SNA: Social Network Analysis
SocNetV: Social Network Visualiser
SQL: Structured Query Language
SSA: Space Syntax Analysis
SVG: Scalable Vector Graphics
TCLTK: Tool Command Language Tool Kit
TIGER: Topologically Integrated Geographic Encoding and Referencing
VB.NET: Visual Basic .NET
VCT: IDRISI Vector Format
visone: Visual Social Networks
VPF: Vector Product Format
VRT: Virtual Datasource
WFS: Web Feature Service
XML: eXtensible Markup Language
Plugins and Scripts Developed to Conduct SSA in FOSS4GIS
1. Introduction

Space syntax analysis is a graph theoretical analysis developed by Hillier and Hanson (2003) for the analysis of spatial configurations. Application of graph theory in spatial studies can actually be traced back to the origin of the respective theory. Indeed, it is strange to notice that Euler’s ([1736] 1998) well known essay on the impossibility of finding a solution for “Königsberg’s Seven Bridge Problem”¹, which is assumed to be the first publication about graph theory (see Hopkins and Wilson (2004) and Alexanderson (2006)), is very illustrative for the employment of graph theory in spatial studies (Figure 1 and Figure 2).

Figure 1. Four land areas and seven bridges in the 17th century Königsberg.

Source: Modified from Joachim Bering’s 1613 illustration in Wikimedia (http://commons.wikimedia.org/wiki/File:Koenigsberg,_Map_by_Bering_1613.jpg) (Public domain)

¹ It is said that in the eighteenth century the people of Königsberg used to entertain themselves by trying to find a solution to a famous problem about the city. In the city there were seven bridges over the river Pregel dividing it into four land areas as it is shown in Figure 1 and Figure 2. Königsberg’s Seven Bridge Problem was to find a route to walk around the city by crossing each of the seven bridges exactly once.
Thus, one can easily trace the employment of graph theory in spatial studies back to the origin of the respective theory. Yet, it can be argued that the earlier and systematic applications of graph theory to the analysis of spatial configurations of basic spatial units (BSUs) can actually be first observed within the field of regional science and geography in order to delimit the planning regions by drawing on functional regions (Beyhan, 2010 and 2011b) (Figure 3).

Figure 3. A simple historical sketch for the socio-spatial methods of analysis based on graph theory.
On the other hand, a systematic application of graph theory in social sciences can be first observed in the socio-metric studies initiated by Jacob Moreno in 1930s (Beyhan, 2011c: 216-219). In terms of co-elaboration of spatial setting and social networks, a rough spatial representation of social networks can be first observed during the 1930s in a study conducted by Loomis and Davidson (1939) who display the social relations among the members of a rural community on a cartographic map. In the subsequent years, one may find the earlier considerations or implications of the employment of SNA for the analysis of spatial processes in the innovation diffusion studies of Hägerstrand ([1953], 1967), though it was not employed by him in an exhaustive way. It is not difficult to infer from his book on “Innovation Diffusion as a Spatial Process” that Hägerstrand was very much influenced by Moreno, founder father of sociometric studies that have in later years become known as SNA.

Nowadays, the interaction between spatial configurations and social networks is increasingly becoming an important field of research both in geography and regional science. As noted above, earlier considerations for the co-elaboration of social networks and spatial configurations can easily be found in Hägerstrand’s pionnerring studies on the spatial diffusion of innovation. Although Hägerstrand’s achievements are revolutionary in terms of examination of diffusion of innovation as a spatial process by drawing on the interactions between different levels and geographical coverage of social networks, the overwhelming importance assigned by him to the metric distance and information networks are criticized by some scholars such as Blaut (1977).²

Indeed, Hägerstrand could not foresee the possibility of implementing the same method of analysis for the elaboration of the spatial configurations themselves and based his analysis on the metric conception of the distance and space as it was the case for majority of studies during the ‘quantitative revolution in geography’ experienced in the 1950s and the 1960s. Reduction of

² Hägerstrand actually recognized the jumping characteristic of innovation diffusion from one location to another, ignoring places in between.
space to distance and probability functions were the inevitable consequences of this revolution. During these years, a systematic application of graph theory in the analysis of spatial structures can also be observed in Haggett and Chorley (1969: 226-257 – location of boundaries) who show how the functional regions can be delimited by employing network analysis.

During the late 1970s, it was Hillier and his colleagues who proposed the employment of graph theory as a method to uncover the spatial configurations themselves without limiting the scope of analysis in terms of metric distance, albeit there are also some problems in the methodological standing offered for Space Syntax Analysis (SSA) by him (see Beyhan (2010) for an elaboration of this particular issue). Today, SNA, SSA and GIS have been actively employed for the analysis of both social networks and spatial configurations.

Particularly, SSA is enthusiastically employed by a range of disciplines including not only urban and regional planning (among many others, see, for example, Asami, Kubat and Istek, 2001; Vaughan, et al., 2005; Choia et al., 2006; Eyüpoğlu, Kubat and Ertekin, 2007; Baran, Rodriguez and Khattak, 2008) and architecture (among many others, see, for example, Bellal, 2004; Dawson, 2008; Aboukhater, 2008; Malhis, 2008), but also especially archeology (Ferguson, 1996; Dawson, 2002; Cutting, 2003; Brusasco, 2004; Clark, 2007) and interdisciplinary search on behavioral and spatial sciences (Peponis et al., 2007; Wineman, Kabo and Davis, 2009).

It is also interesting to note that there is a growing interest in the application of SSA to real estate research (see for example Brown (1999), Matthews and Turnbull (2007), and Enström and Netzell (2008)). By employing SSA and drawing on New Urbanism, Matthews and Turnbull (2007) show that street layout has a significant impact on property value. Enström and Netzell (2008) also show that SSA provides us with important information to understand the intra-urban office rent pattern.

Nevertheless, the number of studies exploring the inter-connections between SNA, SSA and GIS is very limited, which partly stems from the lack of availability of open source graph
theoretic tools for the analysis of spatial configurations. The lack of interconnection is observed not only in the co-elaboration of spatial configurations and social networks, but also in the co-employment of the method and tools developed for the analysis of spatial configurations (such as SSA software programs) and social networks (SNA software programs). It is within this context that SSA Plugins have been developed for the analysis of spatial configurations by building on top of FOSS4GIS and creating some options for the exchange of data between SSA and SNA.

Departing from these introductory remarks, this book will present the process through which SSA plugins have been created for several FOSS4GIS and how to install and use them. However, before proceeding in this direction, it would be helpful to discuss and explain the preference of FOSS4GIS for the development of the plugins. In this respect, the next chapter of the book is devoted to this subject matter. In the third chapter the reader is introduced to the basic concepts and terminology in SSA particularly by drawing on the necessity of interaction between SSA and SNA. After elaboration of basic definitions in SSA, in the fourth chapter of the book structure of SSA scripts are explained by drawing on the algorithms and sample scripts produced for certain parts of SSA. Fifth chapter illustrates how to establish and use the scripting platform available for the creation of plugins in FOSS4GIS for the particular case of SSA Plugins. In the sixth chapter an extended version of installation and user’s guide for SSA Plugin is presented by particularly drawing on the exchange of data between GIS and SNA software programs via SSA plugins. In the final chapter some concluding remarks are presented and discussed via a general evaluation.
2. Why FOSS4GIS for SSA Plugin?

FOSS4GIS is increasingly being made perfect and used by a wider audience thanks to the fact that foundations, local and national governments all over the world financially support the development and employment of FOSS4GIS and motivate the universities to make research in the field because free software grants four freedoms (http://www.fsf.org/):

- The freedom to run the program, for any purpose,
- The freedom to study how the program works, and adapt it to our needs,
- The freedom to redistribute copies so we can help others,
- The freedom to improve the program, and to release our improvements to the public, so that the whole community benefits.

As it is already pointed out by Wineman et al. (2007) and Beyhan (2011a), majority of the software available for space syntax analysis are proprietary that prevents the innovation within the field. After all, “[w]ithout the ability to examine others’ source code, we lose a potential resource for learning and an additional point of verification for peer review” (Wineman et al., 2007).

Although there are also standalone open source SSA software programs such as Syntax2D, they can read and write to a small number of vector or raster formats. Thus, it would be more practical and flexible to build a SSA Plugin on top of existing FOSS4GIS having a wide range of functions including those for reading, writing and editing spatial databases (see Table 1). In this respect, shared libraries used by many FOSS4GIS provide users with a variety of ever developing functions. Modular architecture of the respective software environments further reinforces the creation of more specialized functions in the form of extensions or plugins without duplication of efforts made for the development of a base for a GIS environment.
### Why FOSS4GIS for SSA Plugin?

Table 1. OGR Vector Formats.

<table>
<thead>
<tr>
<th>Format Name</th>
<th>Format Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aeronav FAA files</td>
<td>LIBKML</td>
</tr>
<tr>
<td>ESRI ArcObjects</td>
<td>Mapinfo File</td>
</tr>
<tr>
<td>Arc/Info Binary Coverage</td>
<td>Microstation DGN</td>
</tr>
<tr>
<td>Arc/Info .E00 (ASCII) Coverage</td>
<td>Access MDB (PGeo and Geomedia capable)</td>
</tr>
<tr>
<td>Arc/Info Generate</td>
<td>Memory</td>
</tr>
<tr>
<td>Atlas BNA</td>
<td>MySQL</td>
</tr>
<tr>
<td>AutoCAD DWG</td>
<td>NAS - ALKIS</td>
</tr>
<tr>
<td>AutoCAD DXF</td>
<td>Oracle Spatial</td>
</tr>
<tr>
<td>Comma Separated Value (csv)</td>
<td>ODBC</td>
</tr>
<tr>
<td>CouchDB / GeoCouch</td>
<td>MS SQL Spatial</td>
</tr>
<tr>
<td>DODS/OPeNDAP</td>
<td>Open Document Spreadsheet (ods)</td>
</tr>
<tr>
<td>EDIGEO</td>
<td>OGD1 Vectors (VPF, VMAP, DCW)</td>
</tr>
<tr>
<td>ElasticSearch</td>
<td>OpenAir</td>
</tr>
<tr>
<td>ESRI FileGDB</td>
<td>PCI Geomatics Database File</td>
</tr>
<tr>
<td>ESRI Personal GeoDatabase</td>
<td>Geospatial PDF</td>
</tr>
<tr>
<td>ESRI ArcSDE</td>
<td>PDS</td>
</tr>
<tr>
<td>ESRI Shapefile</td>
<td>PGDump</td>
</tr>
<tr>
<td>FMEObjects Gateway</td>
<td>PostgreSQL/PostGIS</td>
</tr>
<tr>
<td>GeoJSON</td>
<td>EPIInfo .REC</td>
</tr>
<tr>
<td>Géoconcept Export</td>
<td>S-57 (ENC)</td>
</tr>
<tr>
<td>Geomedia .mdb</td>
<td>SDTS</td>
</tr>
<tr>
<td>GeoRSS</td>
<td>SEG-P1 / UKOOA P1/90</td>
</tr>
<tr>
<td>Google Fusion Tables</td>
<td>SEG-Y</td>
</tr>
<tr>
<td>GML</td>
<td>Norwegian SOSI Standard</td>
</tr>
<tr>
<td>GMT</td>
<td>SQLite/SpatiaLite</td>
</tr>
<tr>
<td>GPSBabel</td>
<td>SUA</td>
</tr>
<tr>
<td>GPX</td>
<td>SVG</td>
</tr>
<tr>
<td>GRASS</td>
<td>UK .NTF</td>
</tr>
<tr>
<td>GPSTrackMaker (.gtm, .gtz)</td>
<td>U.S. Census TIGER/Line</td>
</tr>
<tr>
<td>Hydrographic Transfer Format</td>
<td>VFK data</td>
</tr>
<tr>
<td>Idrisi Vector (.VCT)</td>
<td>VRT - Virtual Datasource</td>
</tr>
<tr>
<td>Informix DataBlade</td>
<td>OGC WFS (Web Feature Service)</td>
</tr>
<tr>
<td>INTERLIS</td>
<td>MS Excel format</td>
</tr>
<tr>
<td>INGRES</td>
<td>MS Office Open XML spreadsheet</td>
</tr>
<tr>
<td>KML</td>
<td>X-Plane/Flightgear aeronautical data</td>
</tr>
</tbody>
</table>

Plugins and Scripts Developed to Conduct SSA in FOSS4GIS

There are also some plugins and extensions created in order to conduct SSA in proprietary GIS software (such as \textit{Axwomen} by Bin Jiang (1998) and \textit{AXess 1.0} by Jennifer Brisbane (2006) for ArcInfo, \textit{Confeego} by the commercial company Space Syntax Limited (Gil, Stutz and Chiaradia, 2007) and \textit{Place Syntax Tool} by Ståhle, Marcus and Karlström (2007) for MapInfo, and \textit{Spatialist} by Peponis, Wineman, Rashid, Kim and Bafna (1997) for MicroStation).

As it is remarked by Beyhan (2011a), all these plugins run in a GIS software program that is not free and open source, albeit the respective plugins themselves can be freely available for academic research purposes. Except for the script described by Wang and Liao (2007) and run in GRASS, there was actually no option to conduct SSA in FOSS4GIS. Overall, SSA Plugins described in this book have been built on top of FOSS4GIS because they do not only have a modular architecture facilitating the development of extensions for specific purposes, but also support further developments, customization and open standards owing to their public licence (GNU GPL or MPL) (see Table 2).

Table 2. Scripting opportunities and license types in FOSS4GIS.

<table>
<thead>
<tr>
<th>FOSS4GIS</th>
<th>Type of License</th>
<th>Scripting Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenJUMP</td>
<td>GNU GPL</td>
<td>Java, BeanShell, Jython</td>
</tr>
<tr>
<td>OrbisGIS</td>
<td>GNU GPL</td>
<td>Java, BeanShell</td>
</tr>
<tr>
<td>gvSIG</td>
<td>GNU GPL</td>
<td>Java, Jython, BeanShell, Groovy</td>
</tr>
<tr>
<td>MapWindow</td>
<td>MPL</td>
<td>VB.Net and C#</td>
</tr>
<tr>
<td>SAGA(^a)</td>
<td>GNU (L)GPL</td>
<td>MS Visual C++</td>
</tr>
<tr>
<td>QGIS</td>
<td>GNU GPL</td>
<td>Python</td>
</tr>
<tr>
<td>Thuban</td>
<td>GNU GPL</td>
<td>Python</td>
</tr>
<tr>
<td>OpenEV</td>
<td>GNU LGPL</td>
<td>Python</td>
</tr>
<tr>
<td>R-project</td>
<td>GNU GPL</td>
<td>R Script</td>
</tr>
</tbody>
</table>

\(^a\) Although SAGA GUI is licensed under GNU GPL, SAGA API is licensed under GNU LGPL.

Explanations for licences: GPL: General Public License, LGPL: Lesser GPL, MPL: Mozilla Public License.
Why FOSS4GIS for SSA Plugin?

A general evaluation of advantages and disadvantages of FOSS and proprietary software reveals that although proprietary software has certain advantages in terms of documentation and components’ harmony with each other, it is much more difficult to customize the respective software programs because of their proprietary structure (Table 3). Most of the time proprietary software is not also free and user support depends upon the existence of the software company. Whereas in FOSS we have the freedom to run the program, for any purpose, to redistribute copies and to study how the program works and to change it to make it do what we wish, which opens the door for unlimited customization of FOSS programs. The only negative aspect of FOSS is the fact they usually require a certain level of expertise for the installation of the software and they have an over modular structure which may create some problems for the end users having no training.

Nevertheless, it is this modular structure of FOSS that also makes their customization easier. In addition to the modular architecture of FOSS4GIS, they are also based on ever developing shared libraries thanks to the contribution of volunteers and the support of some foundations and institutions. Once a library is created by a team in a specific scripting language, it is usually ported by the members of other teams to other scripting languages. For example, Java Topology Suite (JTS) originally developed for JUMP (Java) project has been ported to C++ under the name of GEOS (Geometry Engine – Open Source).\(^3\)

JTS is a software library for OpenGIS geometries and methods (spatial predicate functions and spatial operators) such as intersects, touches, buffer, union, polygon building, indexing, and validity. As being a C++ port of JTS, GEOS is intended to contain the complete functionality of JTS in C++. SSA Plugins heavily draw on JTS and GEOS library for the construction of adjusted graph in OpenJUMP, gvSIG, OrbisGIS and Quantum GIS. MapWindow GIS and SAGA also make use of GEOS. Other important shared libraries in C++ can be listed as below;

\(^3\) There is also a .Net port of JTS and it is known as NTS (Net Topology Suite).
• GDAL (Geospatial Data Abstraction Library); raster format reader and writer,

• OGR (OpenGIS Simple Features Reference Implementation); vector format reader and writer (see Table 1 for the list of vector files that can be read via OGR provided that required libraries are loaded into the operating system),

• Proj4 (Cartographic Projections Library); reprojection.\textsuperscript{4}

MapWindow GIS, SAGA, Quantum GIS and OpenEV draw on the above shared libraries in order to read and write raster and vector data formats, and also for coordinate reprojection. In Thuban using Proj4 there is also a support for (testing) OGR. Parallel to GDAL and OGR’s functionality in C++, as a Java GIS toolkit, GeoTools serves data using various OGC specifications and it supports for common geospatial data formats in Java.

As it is noted above, one of the most important characteristics of FOSS4GIS is their modular architecture, which is actually made possible by the easy development of plugins for the respective environment. In this respect it is important to notice that there is also some scripting or console applications in many FOSS4GIS that can be employed for the creation of plugins or extensions (see Table 2). As it will be discussed in a detailed context in Chapter 5, it is possible to develop plugins in many FOSS4GIS by using various scripting languages including Java, BeanShell, Jython, Groovy, Python, C++, C# and Visual Basic.\textsuperscript{5}

Compared with property software there is almost no initial cost of scripting in FOSS4GIS thanks to the fact that majority of the tools and software programs employed in the development of FOSS4GIS is also FOSS. Although some of the tools and software programs employed for the development of certain FOSS4GIS is not FOSS, they are usually free or shareware. The main cost involved in the development of scripts for FOSS4GIS is the time and money required for the training of the people involved in the process.

\textsuperscript{4} There is also a .Net port of Proj4 and it is known as Proj.Net
\textsuperscript{5} BeanShell and Groovy are dynamic scripting languages and tools for Java.
Table 3. Advantages and disadvantages of FOSS versus proprietary software.

<table>
<thead>
<tr>
<th></th>
<th>Proprietary Software</th>
<th>FOSS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Advantages</td>
<td>• developer guarantees that all the components of the software will work in accordance with each other,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• they are usually well documented software,</td>
<td>• freedom to run the program, for any purpose (no license fees),</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• freedom to redistribute copies (unrestricted use),</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• freedom to study how the program works and to change it to make it do what we wish (unlimited customization),</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• support of open standards,</td>
</tr>
<tr>
<td>Disadvantages</td>
<td>• software and maintenance costs,</td>
<td>• they require a certain level of expertise for the installation of the software,</td>
</tr>
<tr>
<td></td>
<td>• difficulty of creating customized applications because of the proprietary structure,</td>
<td>• they have an over modular structure which may create some problems for the end user without training,</td>
</tr>
<tr>
<td></td>
<td>• they are supported only as long as software company exists,</td>
<td></td>
</tr>
</tbody>
</table>

3. Basic Definitions in Space Syntax Analysis and Need for Interaction between SSA and SNA

In this chapter, the reader is provided with the basic terminology about SSA in connection with the need to exchange data between SSA and SNA software programs. Accordingly, the reader is first introduced to the concept of ‘adjusted graph’ and then standard parameters used in SSA are elaborated in connection with SNA. The last two subsections deal with the options available for the construction of line networks and also necessity of interaction between SSA Plugin and SNA software programs.

3.1. What is Adjusted Graph?

As a graph theoretical method of analysis, in SSA the urban space is described by using an adjusted graph in which lines (such as axial lines constructed according to visibility pattern available along streets, segments of streets or streets themselves) are represented as nodes, and the interconnections between them are shown as edges linking the nodes (see Figure 4, Figure 5 and Figure 6). In many respects, construction of the adjusted graph is one of the central parts of software programs designed to perform space syntax analysis.

![Example for a line network](image)

Figure 4. Spatial configuration of street segments in Siteler.
Basic Definitions in SSA and Interaction between SSA and SNA

Figure 5. Street segments at the north-west of Siteler.

Figure 6. Adjusted graph created for the street segments located at the north-west of Siteler.
3.2. Parameters Used in SSA in Connection with SNA

In SSA, as Moody (2000) notes, “a social network is an analogy relating actors and relations to points and lines”. In this analogy, graph theory is used both to analyze and to represent social systems. Any graph in SNA is composed of nodes corresponding to agents in the network and lines corresponding to relations between the respective agents. In similar veins, in SSA the urban space is described by using an adjusted graph in which basic spatial units (BSUs) (such as streets or segments of streets) are represented as nodes, and the interconnections between them are shown as edges linking nodes as it is discussed in the previous section. It should be noted that as they employ the same mathematical infrastructure, graph theory, most of the parameters developed for SNA and SSA actually overlap with each other.

For example, degree centrality which is based on the simple counting of the ties incoming or outgoing from a node in a network is called connectivity in SSA studies (see Equation 1). Since it is a local measure, as Tomko et al. (2008: 44) emphasize, degree centrality does not seem to be a prominent measure for large areas (see particularly Freeman (1977 and 1978) for an elaboration of the concept of centrality). Other forms of centrality measures used in SNA are not formally available in standard space syntax software programs. Nevertheless, a closer examination unveils that ‘mean depth’ (see Equation 4) in SSA actually corresponds to reciprocal value of ‘closeness centrality’ in SNA.

Indeed, one can easily notice that ‘closeness centrality’ of a node can be defined as the reciprocal of ‘mean depth’ which is the average length of the shortest paths between the respective node and every other node. In this respect, as Kalamaras (2010) notes, ‘closeness centrality’ can be “interpreted as the ability to access information through the ‘grapevine’ of network members” because nodes characterized with high closeness centrality have low average length of the path to all the other nodes in the network. Overall, the formulas for the graph theoretic parameters of SSA in connection with SNA can be given as below (see Hillier and Hanson (2003) for the further explanation of equations);
Basic Definitions in SSA and Interaction between SSA and SNA

\[ c_i = k \]  \hspace{1cm} (1)

where \( c_i \) is the connectivity value (degree centrality) for basic spatial unit (BSU) \( i \), \( k \) is the number of other BSUs connected to BSU \( i \);

\[ ctr_i = \sum_{j=1}^{m} \frac{1}{c_j} \]  \hspace{1cm} (2)

where \( ctr_i \) is the control value for BSU \( i \), \( m \) is the number of neighbors of BSU \( i \), and \( c_j \) is the connectivity value of BSU \( j \);

\[ td_i = \sum_{j=1}^{n} d_{ij} \]  \hspace{1cm} (3)

where \( td_i \) is the total depth (farness) value for BSU \( i \), \( d_{ij} \) is the geodesic distance between BSU \( i \) and BSU \( j \), and \( n \) is the total number of BSUs;

\[ md_i = \frac{td_i}{n-1} \]  \hspace{1cm} (4)

where \( md_i \) is the mean depth value (reciprocal value of ‘closeness centrality’) for BSU \( i \), \( td_i \) is the total depth value for BSU \( i \), and \( n \) is the total number of BSUs;

\[ g_i = \left( \frac{2(md_i - 1)}{n-2} - \frac{2(n \log_2((n+2)/3)-1+1)}{(n-1)(n-2)} \right)^{-1} \]  \hspace{1cm} (5)

where \( g_i \) is the global integration value for BSU \( i \), \( md_i \) is the mean depth value for BSU \( i \), and \( n \) is the total number of BSUs.
Although SSA parameters such as connectivity, ‘total depth’ (see Equation 3), and mean depth directly or indirectly correspond to some of the parameters already available in SNA, this is not the case for all the parameters developed for SSA studies. Nevertheless, some parameters developed for SNA provide approximate measures for the respective parameters used in SSA studies. For example, it is well documented in some studies that ‘closeness centrality’ highly correlates with the global integration values. Theoretically, BSUs associated with high ‘closeness centrality’ and ‘global integration’ values reveal the core of the network.

In SSA, two other important parameters are ‘local depth’ and ‘local integration’ values calculated for each BSU by taking into account the network within a given radius. Local integration is basically employed to unveil the variation of integration across the network within a range of generally three closest BSUs. Actually, both ‘local depth’ and ‘local integration’ values reflect the average length of pedestrian walks, which has been proven empirically by many SSA studies (among many others see Liu (2007) and Hillier et al. (1993)). Although Tomko et al. (2008: 44) consider local integration as “a measure designed to be applied to axial map analysis, as opposed to named streets”, it is clear that in areas with relatively short and straight streets, the respective parameter can be employed in order to observe the relationship between the pedestrian circulation and the spatial configuration of BSUs composed of streets.

It is notable that there is no direct corresponding parameter for local integration value in SNA. Nevertheless, it is also important to note that some of the critical parameters (such as ‘betweenness centrality’) available in SNA are not available in a standard SSA software program. Betweenness centrality (Equation 6) of a node is the ratio of all geodesics between pairs of nodes which run through the respective node. As Kalamaras (2010) notes, it reflects how often a node lies on the geodesics between the other nodes of the network. As betweenness centrality “provides the

---

6 Total depth in SSA actually corresponds to farness parameter calculated for closeness centrality in SNA.
means to quantify the likelihood a graph node will lie on a shortest path between two other nodes of the graph”, it actually “provides a global value for a specific network element” such as segments in a street (Tomko et al., 2008: 44). As Tomko et al. (2008: 44) argue betweenness centrality can be “expected to reflect the probability of being selected by a frequent wayfinder”.

Betweenness centrality of a node $i$, $C^B_i$, is defined by Tomko et al. (2008: 45) with reference to Freeman (1977) as below:

$$C^B_i = \sum_{i \neq j \neq k} \frac{n^j_{k(i)}}{n^j_{k}}$$

where $i$, $j$ and $k$ are distinct BSUs, $n^j_{k}$ shows the number of shortest paths between BSU $j$ and BSU $k$, $n^j_{k(i)}$ denotes the number of such paths leading through BSU $i$.

Considering the lack of interaction between SSA and SNA, SSA Plugins are designed to export adjusted graph into a SNA network file format for further analysis of the spatial configuration concerned in SNA software programs.

### 3.3. Options for the Construction of Line Networks

As pointed out earlier, SSA Plugins are particularly designed to be operational for line networks (such as streets) in GIS. In the construction of a line network for the representation of a spatial configuration there are several alternatives based on the questions that will be addressed in a study. One can roughly differentiate between segment approach, axial lines and named streets (Hillier and Hanson, 2003, Jiang and Claramunt, 2004; Tomko et al., 2008; Beyhan, 2010). Thanks to the modules and built-in functions available in FOSS4GIS to process geographic data, any spatial configuration consisting of lines can also be rearranged, for example, according to the continuity between contiguous lines regarding the angle between them.
In the first option (segment approach) available for the construction of line networks representing, for example, streets, streets are divided into their constituent parts by producing different nodes for each segment involved in any street. Respective segments are defined according to the intersection between the streets such that each segment can be distinguished by its end points that are either connected to other segments or dead ends without any connection to other segments (Beyhan, 2010). In this way, each segment is treated separately in the space syntax analysis (for an example, see Demšar, Špatenková and Virrantaus (2007) who take “the original vertices and edges of the street network as the basic spatial elements in the calculation of the line graph”).

The second option which employs axial lines is more traditional in SSA studies and actually based on the visibility pattern of a given spatial configuration of, for example, streets. In this option, line network is constructed upon the vistas. Accordingly, any given spatial configuration is partitioned into axial lines in such a way that as long as possible the respective configuration is represented by a set of longest but fewest lines along which a set of fattest convex spaces can be covered in terms of the area visible by both ends of the respective lines. The concept of axial line was first introduced by Hillier and Hanson (2003).

The third and last option takes the streets themselves as the basic spatial unit of analysis without disaggregating them into segments or axial lines (for the first experiments in this direction see Jiang and Claramunt (2004), who employ the streets as the BSUs in the construction of adjusted graph). According to the third option, for each separate street in a given area a node is created in the adjusted graph according to the names of the respective streets. In a recent study (Tomko et al., 2008: 44), this approach is labeled as ‘named streets’ approach. Named streets approach puts its emphasis on the fact that in daily life people usually refer to streets by only employing their names not the particular segments that actually constitute the respective streets.

Although it can be argued that this reflects some part of the reality, there is no doubt that under certain circumstances street
names can not be taken as the building block of a space syntax analysis. Especially, in an area where the length of streets are longer than as usual or expected, one can not refer to a specific point placed along a very long street by simply providing the name of the respective street. Furthermore, segments of the streets are also part of the common knowledge if it is practical to name them according to their position in the main street (such as middle, north, south, east or west part of a very long street).

3.4. Necessity of Interaction between SSA Plugin and SNA Software Programs

Standard parameters available for SSA overlap only with a small segment of the parameters available in many SNA software programs, albeit they can also be used to understand the nature of spatial configurations. Thus, from the very beginning of SSA Plugin project there was an intention to create an interaction between SSA and SNA software programs to cross-fertilize the efforts made for them. Instead of producing scripts in FOSS4GIS for a series of graph theoretic parameters (such as other measures of centrality including betweenness) that can easily be calculated in many SNA software programs, I thought that it would be wise to create an option for the transfer of adjusted graph produced by using SSA Plugin for further analysis of the spatial configuration concerned in the already available SNA software programs. As a result of this consideration, two of the mostly widely used SNA file formats have been chosen to produce network files from the adjusted graph constructed by using SSA Plugin (see Table 4).

The first one is Ucinet’s Data Language (“.dl”) file format (Borgatti, Everett and Freeman, 2002; Hanneman and Riddle, 2005: 72-73) and the second one is Pajek’s Network (“.net”) file format (Batagelj and Mrvar, 2011: 73-76). Both of the respective file formats actually store edge list in ASCII format together with some header information and it is always possible to edit them by using a standard text editor in case a designated SNA could not open any of them. A sample network file written in “.net” and “.dl” file formats can be seen in Table 4 (see Figure 7 for the graph produced by using the network information in Table 4).
Table 4. SNA file formats chosen for SSA Plugin

```
Vertices 8
1 "4817"
2 "4155"
3 "5159"
4 "4748"
5 "5083"
6 "4481"
7 "5193"
8 "1594"

Arcs
1 6
1 8
2 3
2 5
3 2
3 4
4 3
4 7
4 8
5 2
6 1
7 4
8 1
8 4
```

```
.Net file format:

(dl n= 8
format = edgelist1
labels embedded
data: 4817 4481
4817 1594
4155 5159
4155 5083
5159 4155
5159 4748
4748 5159
4748 5193
4748 1594
5083 4155
4481 4817
5193 4748
1594 4817
1594 4748)
```

Figure 7. Sample adjusted graph.
4. Conceptualization of SSA in GIS and Structure of SSA Scripts and Plugins

In terms of the spatial databases that can be processed by a standard GIS software program, there are alternative approaches for the graph theoretical analysis of the spatial configurations by using SSA (Table 5). The first option is to employ vector layers as the base to represent any spatial configuration such streets, buildings or units of a building. The second option is to use image layers (particularly grid images) as the base for the representation of the spatial configurations. There are certain advantages and disadvantages of both options.

Table 5. Different options that may be followed in FOSS4GIS for conducting SSA.

<table>
<thead>
<tr>
<th>studies based on</th>
<th>options</th>
</tr>
</thead>
</table>
| vector databases | • **polygon networks** (convex spaces - Hillier and Hanson (2003)) (polygons are created to form convex spaces and final map is composed of contiguous areas),  
• **line networks** (as discussed before there are three different options in the construction of line networks; (1) axial (based on vistas), (2) segment and (3) named line models) (Hillier and Hanson, 2003; Jiang and Claramunt, 2004; Tomko et al., 2008; Beyhan, 2010). |
| grid databases   | • **grid isovist analysis** based on the visibility of areas from a single point,  
• creating **visibility index** for each cell in an area,  
• application of SSA and SNA tools and parameters to visibility network. |
For both options there are also alternative ways of representing a given spatial configuration. In vector layers, existing spatial configuration can be represented as an interconnected set of either convex spaces (such as the rooms, corridors and other spatial units in a building) or lines (such as the streets and axial lines for a given urban fabric). A given set of point distribution can also be represented as an interconnected set of spaces by creating conditions of adjacency between the points in the respective distribution (such as the creation of voronoi diagrams and neighborhood matrices according to the inclusion of a point within the circle drawn around another point).

Although the plugins described in this book is operational on vector layers for the analysis of particularly line networks, a space syntactic analysis of the convex spaces can also be realized in some FOSS4GIS owing to the built-in functions available in the respective FOSS4GIS in terms of the scripting infrastructure they have. In this respect, core of SSA script is composed of three basic sections (Figure 8). Accordingly, the first section of the script is devoted to the construction of adjusted graph for the spatial configuration that will be analyzed. Adjusted graph actually shows the adjacency between BSUs and it can easily be constructed by checking whether or not two features (BSUs) intersect each other. As in some FOSS4GIS there is a built-in function for the respective task, the construction of adjusted graph in them is easier compared with others lacking such kind of a built-in function.

The second section of the plugin calculates geodesics between BSUs involved in the spatial configuration on the base of the adjusted graph produced in the first section. As it is explained in the following part of this chapter, calculation of geodesics is remarkably accelerated by improving the algorithm employed for this purpose throughout the process devoted to the creation of SSA Plugins. In the last section of the main script, the standard space syntax parameters are calculated on the base of the geodesics calculated in the second part of the script.
In addition to these basic sections, in SSA Plugin there are also scripts used for GUI (dialog box), exporting adjusted graph into a SNA file format and function (Pearson Correlation) employed in the calculation of intelligibility value. They can also be considered as other sections of the overall script used in SSA Plugin. Each section depends upon the other ones in several different ways. For example, not only adjacency matrix but also list of list used in the calculation of geodesics in the second section of the main script is actually constructed in the first section of the main script devoted to the construction of adjusted graph.

In a similar fashion, before the calculation of parameters, attribute table of the layer covering spatial configuration should also be processed to add new columns in order to write the parameters to
the table. If we choose an option to export adjusted graph or geodesic distance into an external file, the scripts responsible for these tasks are also integrated into other sections in order to benefit from the routines available in the sections concerned instead of creating new routines. In the following parts of this chapter, each section of the plugin is explained by presenting the algorithm and sample scripts used in the respective part.

### 4.1. Construction of Adjusted Graph

Although in some FOSS4GIS (such as OpenJUMP, QGIS, gvSIG and OrbisGIS) there are some built-in functions in order to check whether or not two features intersect, in some others such kind of built-in functions are not available and it is necessary to produce scripts to check the intersection between features from scratch. After numerous tests, it is also observed that in some FOSS4GIS having a built-in function for checking the intersection between features, the respective function is not properly functioning. For the respective FOSS4GIS this has been reported as bugs for the communities concerned and the intersection of features are checked by employing SSA sub-scripts particularly produced for this purpose.

If the intersection of features could not be checked by using a built-in function, it is always possible to determine whether or not geographic features intersect by using bounding boxes (BB) and segments of the features concerned. As it is remarked by Beyhan (2011a), instead of checking intersection of each segment of a feature with any segment of another feature, it would be wise first to check whether or not their respective bounding boxes overlap each other. If BBs of two features do not overlap each other, it will be redundant to conduct a segment vise checking for the intersection of the respective features. Thus, any function available for the determination of bounding box (BB) gains critical importance. If no function is available for this, BBs can also be easily calculated by retrieving the coordinates of the points placed along a line feature. After calculation of BBs, if BBs of two features overlap with each other, a segment vise checking is employed in order to ascertain whether or not two features intersect.
In this section, firstly the construction of adjusted graph for those FOSS4GIS having a built-in function for checking the intersection between features is illustrated by drawing on the sample scripts produced for this purpose. Subsequently, if there is no built-in function for the determination of intersection, it is also shown how to retrieve-calculate BBs and check overlap between BBs. Lastly, checking intersection between line segments is illustrated by again drawing on sample scripts. The algorithm employed in some SSA Plugins created for FOSS4GIS having no built-in function for checking the intersection between features involved in a layer is based on the assumption that user will explode multi-line features into their constituent parts before they use SSA Plugin. Thus, there is no support for multi-line features in some SSA Plugins created for FOSS4GIS having no built-in functions for checking the intersection between features. If there is a multi-line feature in a vector layer, in the respective FOSS4GIS SSA Plugin only takes into account the first part of the respective feature without iterating over other parts for the construction of adjusted graph.

4.1.1. Using Built-in Functions to Check Intersection

In OpenJUMP, gvSIG, OrbisGIS and Quantum GIS, there are built-in functions to check the intersection between the features involved in a vector layer. For this purpose, features in a vector layer should be iterated by using two loops. First loop is employed in order to select a feature in a sequential manner and the second loop is used in order to check whether the respective feature intersect with other features in the layer:

```
katman = current layer
sna[][] is an two dimensional integer array
for f = 1 to number of features in katman
    o1 = f\text{th} feature in katman
    for s = 1 to number of features in katman
        o2 = s\text{th} feature in katman
        if o1 intersects o2 then sna[f][s] = 1
    next s
next f
```
For the above pseudo-code, \( sna \) stores the adjusted graph produced for the selected vector layer. And below is a sample script (Script 1) produced for the construction of adjusted graph in OpenJUMP BeanShell;

**Script 1. Construction of adjusted graph in OpenJUMP BeanShell.**

```java
layer = wc.layerNamePanel.selectedLayers[0];
fc = layer.featureCollectionWrapper;
int xx;
int yy;
int obj = fc.size();
int[][] sna = new int[obj][obj];
xx = obj;

for (f : fc.features) {
    cc = f.getGeometry();
    yy = obj;
    for (g : fc.features) {
        dd = g.getGeometry();
        ser = cc.intersects(dd);
        if ((ser == true) && (xx != yy)) {
            sna[obj-xx][obj-yy] = 1;
        }
    }
    yy--;
}
xx--;
```

If FOSS4GIS selected to develop SSA Plugin does not have any built-in function for checking the intersection of geographic features, we will first need to calculate bounding boxes or use the built-in function available for this purpose in order to determine whether or not BBs of two features overlap each other.

### 4.1.2. Calculation of Bounding Boxes

Once we have all the coordinates of points placed along a feature, bounding box for the respective feature can be calculated by using the available functions in the scripting language used to create plugins. Below there is a sample script produced for the calculation of BBs in Python for OpenEV (Script 2). The first loop \( s1 \) iterates over the features in the selected vector layer and
the second loop \( p_1 \) iterates over the coordinates of the selected feature in order to calculate BB of the respective feature. Accordingly, at the end of the second loop, we have \( xmax, xmin, ymax \) and \( ymin \) of BB of the selected feature. The third and fourth loops do the same calculations for the features that will be checked for whether or not they intersect with the features selected in the first loop.

**Script 2. Calculation of bounding boxes in OpenEV.**

```python
for s1 in range(obj):
    i1 = shapes[s1]
    row1 = i1.get_nodes()
    e1 = i1.get_node(0)
    xmax = e1[0]
    ymax = e1[1]
    xmin = e1[0]
    ymin = e1[1]

    for p1 in range(row1-1):
        e1 = i1.get_node(p1+1)
        x1 = e1[0]
        y1 = e1[1]
        if (x1 > xmax): xmax = x1
        if (x1 < xmin): xmin = x1
        if (y1 > ymax): ymax = y1
        if (y1 < ymin): ymin = y1

    for s2 in range(obj):
        i2 = shapes[s2]
        row2 = i2.get_nodes()
        e2 = i2.get_node(0)
        umax = e2[0]
        vmax = e2[1]
        umin = e2[0]
        vmin = e2[1]

        for p2 in range(row2-1):
            e2 = i2.get_node(p2+1)
            x2 = e2[0]
            y2 = e2[1]
            if (x2 > umax): umax = x2
            if (x2 < umin): umin = x2
            if (y2 > vmax): vmax = y2
            if (y2 < vmin): vmin = y2
```

27
Plugins and Scripts Developed to Conduct SSA in FOSS4GIS

4.1.3. Checking Overlapping between Bounding Boxes

After calculation of BBs of two features, the overlapping between them can easily be checked, for example, by simply employing the following Python script (Script 3) in OpenEV;

Script 3. Checking overlapping between bounding boxes in OpenEV.

```python
if (xmax<umin or ymin>vmax): bonuc = "0"
elif (xmin>umax or ymin>vmax): bonuc = "0"
elif (xmax<umin or ymax<vmin): bonuc = "0"
elif (xmin>umax or ymax<vmin): bonuc = "0"
elif s1==s2: bonuc = "0"
else: bonuc = "1"
```

where `bonuc` stands for overlapping between BBs of two features. If it is equal to 1, it means that the respective BBs overlap. In some FOSS4GIS there is some built-in functions in order to calculate BBs. For example, in Thuban there is such kind of a function (`ShapesBoundingBox`) that reduces the amount of script we produced in order to test whether or not BBs of two features overlap (see Script 4)

Script 4. Checking overlapping between BBs in Thuban.

```python
for s1 in self.layer.ShapeStore().AllShapes():
    i1 = s1.ShapeID()
    k1 = s1.Points()
    l1 = k1[0]
    row1 = len(l1)
    bbox = self.layer.ShapesBoundingBox([i1])
    bonuc = "0"

for s2 in self.layer.ShapeStore().AllShapes():
    i2 = s2.ShapeID()
    k2 = s2.Points()
    l2 = k2[0]
    row2 = len(l2)
    sbox = self.layer.ShapesBoundingBox([i2])

    if (bbox[2]<sbox[0] or bbox[1]>sbox[3]): bonuc="0"
    elif (bbox[0]>sbox[2] or bbox[1]>sbox[3]): bonuc="0"
    elif (bbox[2]<sbox[0] or bbox[3]<sbox[1]): bonuc="0"
    elif (bbox[0]>sbox[2] or bbox[3]<sbox[1]): bonuc="0"
    elif i1==i2: bonuc="0"
    else: bonuc = "1"
```
4.1.4. Checking Intersection between Line Segments

If BBs of two features overlap with each other, a segment based checking of intersection between features can be done in order to identify whether or not two features intersect or contiguous to each other. For this purpose simple equation for a straight line can be used. If it is considered that the equation for any line segment placed along a polyline can be written as;

\[ v = a + bu \]  

(7)

where \( v \) and \( u \) stand for, respectively, vertical and horizontal coordinates, \( b \) is the slope of the line and \( a \) is a constant, the intersection between two line features can easily be calculated by iterating over the segments placed in the respective polyline features. It is known that if the coordinates of two points (point 1 and point 2) placed along a line are known, then;

\[ \begin{align*}
1 & \quad 2 \\
1 & \quad 2 \\
u & = u - u \\
v & = v - v \\
b & = \frac{v_2 - v_1}{u_2 - u_1} \\
\end{align*} \]  

(8)

If equation (7) is solved for one of the points (let it be the first point), then;

\[ a = v_1 - bu_1 \]  

(9)

Based on these equations, the intersection point for two line segments of different polyline features can easily be calculated by solving the following equation first for \( u \), and then for \( v \);

\[ a_1 + b_1u = a_2 + b_2u \]  

(10)

where subscripts 1 and 2 placed after \( a \) and \( b \) stand for, respectively, the first and the second line segments. Equation (10) can also be rewritten as following;

\[ u = \frac{a_1 - a_2}{b_1 - b_2} \]  

(11)

If equation (7) is solved for \( v \) for one of the segments (let it be the first segment), then;
\[ v = a_i + b_i u \]  \hspace{1cm} (12)

Since the intersection point is actually calculated for the equations representing continuous lines, it may not lie along the actual segments for which calculations are made. Hence, the intersection point should be checked for whether or not it lies along the actual segments. If the respective point satisfies the following conditions, it means that it lies along the actual segments and two line features intersect with each other;

\[
(u_1 - u)(u - u_2) \geq 0 \quad \text{and} \quad (x_1 - x)(x - x_2) \geq 0 \\
(v_1 - v)(v - v_2) \geq 0 \quad \text{and} \quad (y_1 - y)(y - y_2) \geq 0
\]

where subscripts 1 and 2 stand for, respectively, the end points of the first (\( u \) and \( v \)) and the second (\( x \) and \( y \)) segments.

As it is remarked by Beyhan (2011a), special treatments are also required for the cases where any of the two segments lies along a horizontal line for which the calculation of \( b \) gives a “division by zero” error.

The procedure described above for checking the intersection between line features can be written in Python for Thuban as shown in Script 5. Except for the functions used to iterate over the segments and nodes located along a polyline feature, Script 5 can also be used in OpenEV. For example, although in Thuban we can iterate over nodes located along a polyline feature by simply calling the items of an array, in OpenEV we should use \texttt{get_node} in order to iterate over the nodes.

Script 5. Checking intersection between line features in Thuban.

```python
if bonuc == "1":

    for p1 in range(row1-1):
        p11 = l1[p1]
        p12 = l1[p1+1]
        x1 = p11[0]
        y1 = p11[1]
        x2 = p12[0]
        y2 = p12[1]
```
Conceptualization of SSA in GIS and Structure of SSA Scripts

\[ k = 0 \]
\[ \text{if} \ (x_2 \ == \ x_1): \ k = 1 \]
\[ \text{else}: \]
\[ b_1 = (y_2 - y_1)/(x_2 - x_1) \]
\[ a_1 = y_1 - b_1*x_1 \]

for \( p_2 \) in range(row2-1):
\[ p_{21} = 12[p_2] \]
\[ p_{22} = 12[p_2+1] \]
\[ u_1 = p_{21}[0] \]
\[ v_1 = p_{21}[1] \]
\[ u_2 = p_{22}[0] \]
\[ v_2 = p_{22}[1] \]

\[ l = 0 \]
\[ \text{if} \ (u_2 == u_1): \ l = 1 \]
\[ \text{else}: \]
\[ b_2 = (v_2 - v_1)/(u_2 - u_1) \]
\[ a_2 = v_1 - b_2*u_1 \]

if ((k == 0) and (l == 0)):
\[ \text{if} \ (b_1 == b_2): \ \text{sonuc} = "0" \]
\[ \text{else}: \]
\[ x_i = 0 - (a_1-a_2)/(b_1-b_2) \]
\[ y_i = a_1 + b_1*x_i \]
\[ \text{if} \ ((x_1-x_i)*(x_i-x_2)>=0 \ \text{and} \ (u_1-x_i)*(x_i- \]
\[ u_2)>=0 \ \text{and} \ (y_1-y_i)*(y_i-y_2)>=0 \ \text{and} \ (v_1- \]
\[ y_i)*(y_i-v_2)>=0): \ \text{sonuc}="1" \]
\[ \text{else}: \ \text{sonuc} = "0" \]

if ((k == 1) and (l == 1)):
\[ \text{sonuc} = "0" \]

if ((k == 1) and (l == 0)):
\[ t_1 = a_2 + b_2*x_1 \]
\[ \text{if} \ ((u_1>=x_1 \ \text{and} \ u_2<=x_1) \ \text{or} \ (u_2>=x_1 \ \text{and} \ \]
\[ u_1<=x_1)) \ \text{and} \ ((y_1>=t_1 \ \text{and} \ y_2<=t_1) \ \text{or} \ \]
\[ (y_2>=t_1 \ \text{and} \ y_1<=t_1)): \ \text{sonuc} = "1" \]
\[ \text{else}: \ \text{sonuc} = "0" \]

if ((k == 0) and (l == 1)):
\[ t_1 = a_1 + b_1*u_1 \]
\[ \text{if} \ ((x_1>=u_1 \ \text{and} \ x_2<=u_1) \ \text{or} \ (x_2>=u_1 \ \text{and} \ \]
\[ x_1<=u_1)) \ \text{and} \ ((v_1>=t_1 \ \text{and} \ v_2<=t_1) \ \text{or} \ \]
\[ (v_2>=t_1 \ \text{and} \ v_1<=t_1)): \ \text{sonuc} = "1" \]
\[ \text{else}: \ \text{sonuc} = "0" \]

if ((x_1==u_1 \ \text{and} \ y_1==v_1) \ \text{or} \ (x_2==u_1 \ \text{and} \ y_2==v_1) \ \text{or} \ (x_1==u_2 \ \text{and} \ y_1==v_2) \ \text{or} \ (x_2==u_2 \ \text{and} \ y_2==v_2)): \ \text{sonuc} = "1"
if (sonuc == "1"): break

if (sonuc == "1"):
    if self.cb_dl.GetValue():
        edges.write(betiket[i1])
        edges.write(" ")
        edges.write(betiket[i2])
        edges.write("\n")
        sna[i1][i2] = 1
        cntr = cntr + 1
        iliski.append(i2)
        break
    sonuc = "0"
    bonuc = "0"

4.2. Calculation of Geodesic Distance

After the construction of the adjusted graph showing the contiguity between the features (BSUs) in a vector layer (spatial configuration), geodesic distance between BSUs can be calculated by developing a script for this purpose. The contiguity between the nodes in a graph can be represented by creating an adjacency matrix where 0 and 1 stand for, respectively, non-adjacent and adjacent nodes indicated by the row and column index of the matrix. As a matter of fact all adjacency matrices are actually square matrices (see matrix (13)).

\[
X_{ij} = \begin{bmatrix}
X_{11} & X_{12} & \cdots & \cdots & \cdots & X_{1(c-1)} & X_{1c} \\
X_{21} & X_{22} & \cdots & \cdots & \cdots & X_{2(c-1)} & X_{2c} \\
\vdots & \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\
\vdots & \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\
\vdots & \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\
X_{(r-1)1} & X_{(r-1)2} & \cdots & \cdots & \cdots & X_{(r-1)(c-1)} & X_{(r-1)c} \\
X_r & X_2 & \cdots & \cdots & \cdots & X_{(c-1)} & X_c
\end{bmatrix} \quad (13)
\]
If it is assumed that the above matrix (13) represents an adjacency matrix with \( r \) rows and \( c \) columns, the geodesics between the nodes indicated by the row and column index of the matrix can be calculated by running the following Python script (directly operational in Quantum GIS, OpenEV and Thuban SSA Plugins and also operational in OpenJUMP Jython and gvSIG Jython SSA plugins with some modification for arrays) where \( sna \) and \( dst \) stand for, respectively, adjacency and geodesic matrices:


```python
dst = sna
for mas in range(obj):
    for kas in range(obj-1):
        fas = 1
        for sas in range(obj):
            if ((dst[mas][sas] == (kas+1)) and (sas != mas)):
                for tas in range(obj):
                    if ((dst[sas][tas] == 1) and (tas != mas)) and ((dst[mas][tas] > (kas+1)) or (dst[mas][tas] == 0)):
                        dst[mas][tas] = kas + 2
                        fas = fas + 1
                    if (fas == 1): break

In order to calculate geodesics between the nodes in a connected graph, algorithmically four nested loops and a simple counter are employed in the script given above. The first \((mas)\) and third \((sas)\) loops functions respectively as the row and column counters required in order to read the values in the adjacency matrix. The second loop \((kas)\) is employed as an accumulator index. If the value of a non-diagonal cell identified by the values of the first and third loops is equal to the sum of 1 and the accumulator index that takes values beginning from 0, another loop \((tas)\) is initiated in order to detect the nodes contiguous to the node which is indicated by the value of the third loop and adjacent to or within a specified distance \((kas + 1)\) from the node specified by the value of the first loop. In other words, the first run of the fourth loop \((tas)\) for each couple of the adjacent nodes specified by the values of \(mas\) and \(sas\) (regularly incremented in
the first and third loops) detects the nodes located within a geodesic of 2 from the node which is identified by the value of the first loop. If such kind of a node is identified, the value of the cell identified by the values of first and fourth loops is set to the value of the accumulator index ($kas$) plus 2. In order to shorten the time period spent for the calculation of geodesics between the node specified by the value of the first loop and the other nodes in the graph, a counter named $fas$ is employed.

After identification of all the nodes located within a geodesic of 2 from the node indicated by the value of $mas$, the value of the second loop is incremented by 1 in order to find the nodes located within a geodesic of 3 from the respective node. If all the geodesics between the first node indicated by the value of $mas$ and the other nodes are calculated, the counter $fas$ breaks the second loop and resets the accumulator index for the calculation of the geodesics between the second node indicated by the value of $mas$ and the other nodes. The respective procedure is repeated till the matrix showing geodesics between the nodes in the graph is constructed. Some conditionals are also used in order to prevent the miscalculations in the algorithm.

In order to increase the speed of the algorithm, the fourth loop is modified by employing a two dimensional array list showing only the nodes adjacent to each other and another one dimensional array list created from the first one and showing only the nodes that are adjacent to the node indicated by the value of the third loop ($sas$). Jython script containing the respective modifications is given below (Script 7);

**Script 7. Calculation of geodesics in Jython.**

```python
for mas in range(obj):
    for kas in range(obj-1):
        fas = 1
        for sas in range(obj):
            if ((dst[mas*obj+sas]==(kas+1)) and (sas!=mas)):
                iliski = fortas[sas]
                sizeOfList = len(iliski)
                for sIndex in range(sizeOfList):
                    tas = iliski[sIndex]
```

if ((tas!=mas)) and 
((dst[mas*obj+tas]>(kas+1)) or 
(dst[mas*obj+tas]==0)):
    dst[mas*obj+tas] = kas + 2
    fas = fas + 1
if (fas == 1): break

In the script given above *fortas* shows the two dimensional array list (list of lists) storing the index values for the adjacent nodes and *iliski* shows the array list indicating only the nodes contiguous to the node specified by the value of the third loop (*sas*).

### 4.3. Calculation of Parameters and Writing Results to Attribute Table

Subsequent to the calculation of geodesics, calculation of standard SSA parameters is actually an easy task in terms of algorithmic difficulty involved in a script. Nevertheless, writing results to the attribute table may involve some difficulties as there is no well written guide for this particular task in most of FOSS4GIS. Except for Thuban and gvSIG, the calculated parameters could be written to the attribute table of the layer selected for the analysis. In gvSIG and Thuban, results of the analysis (parameters) are stored in a new file or layer in the view. Script 8 is a sample OpenJUMP BeanShell script showing how to add new fields to the attribute table of a selected layer, to calculate standard space syntax parameters based on the geodesic distance and degree centrality values calculated in the previous sections of SSA script, and to write the respective parameters to the attribute table.

**Script 8. Calculation of parameters and writing them to attribute table in OpenJUMP BeanShell.**

```java
// adding standard space syntax parameters as new fields to the attribute table;
fs.addAttribute("Lineno", AttributeType.INTEGER);
fs.addAttribute("Connectivity", AttributeType.INTEGER);
fs.addAttribute("TotalDepth", AttributeType.INTEGER);
fs.addAttribute("MeanDepth", AttributeType.DOUBLE);
fs.addAttribute("GlobalInteg", AttributeType.DOUBLE);
fs.addAttribute("LocalDepth", AttributeType.INTEGER);
```
Plugins and Scripts Developed to Conduct SSA in FOSS4GIS

```java
fs.addAttribute("LocalInteg", AttributeType.DOUBLE);
fs.addAttribute("Control", AttributeType.DOUBLE);

newDataset = new FeatureDataset(fs);

xx = obj;

double[] var1 = new double[obj];
double[] var2 = new double[obj];

// calculation of space syntax parameters
for (f : fc.features) {
    nf = new BasicFeature(fs);
    for (i = 0 ; i < (fs.attributeCount-8) ; i++) {
        nf.setAttribute(i, f.getAttribute(i));
    }

    int td = 0;
    double md;
    double ra;
    double rr;
    double gint;
    int locd = 3;
    int deg3 = 0;
    int ld3 = 0;
    double lmd;
    double lra;
    double lrr;
    double lint;
    double say = obj;
    double cntrl = 0;
    int cnt = 0;

    double dvl=((2*(say*((Math.log((say+2)/3)/
Math.log(2))-1)+1))/((say-1)*(say-2)));

    for (int vas = 0; vas < obj; vas++) {
        td = td + dst[obj-xx][vas];
        if (dst[obj-xx][vas] == 1) {
            cnt++;
            double datr = atr[vas];
            cntrl = cntrl + 1/datr;
        }
        if (dst[obj-xx][vas] < locd) {
            ld3 = ld3 + dst[obj-xx][vas];
            deg3++;
        }
    }
}
```
4.4. Calculation of Intelligibility Value

Intelligibility value in SSA is actually Pearson Correlation between connectivity and global integration. Pearson Correlation analysis is employed in order to determine the relationship between two variables. It measures the association between two variables by means of a coefficient that is a value between -1 and 1. The higher the coefficient value (in absolute terms), the more closely the variables associated with each other. The equation (14) for the linear correlation coefficient can be given as below;
where \( \overline{x} \) and \( \overline{y} \) are the means of respectively independent and dependent variables, and \( R \) is Pearson Correlation value.

Above formula can be rewritten as below;

\[
R = \frac{\text{tot\_coproduct}}{\sqrt{\text{tot\_sq\_x} \times \text{tot\_sq\_y}}}
\]  

where;

\[
\text{tot\_coproduct} = \sum_{i=1}^{n} (x - \overline{x}) \times (y - \overline{y})
\]  

\[
\text{tot\_sq\_x} = \sum_{i=1}^{n} (x - \overline{x})^2
\]  

\[
\text{tot\_sq\_y} = \sum_{i=1}^{n} (y - \overline{y})^2
\]

Jython (Script 9) and BeanShell (java) (Script 10) versions of the script produced in order to calculate Pearson Correlation according to the above formulas (equations (15), (16), (17) and (18)) are given below;


```python
def getPearsonCorrelation(var1, var2):
    res = 0
    tsq_x = 0
    tsq_y = 0
    tot_cp = 0
    mean_x = 0
    mean_y = 0
    for i in range(len(var1)):
        mean_x += var1[i] / len(var1)
mean_y += var2[i] / len(var1)
for i in range(len(var1)):
    dlt_x = var1[i]-mean_x
    dlt_y = var2[i]-mean_y
    tsq_x += (dlt_x * dlt_x)
    tsq_y += (dlt_y * dlt_y)
    tot_cp += (dlt_x * dlt_y)
res = tot_cp/(math.sqrt(tsq_x)*math.sqrt(tsq_y))
return result


double result = 0;
double tsq_x = 0;
double tsq_y = 0;
double tot_cp = 0;
double mean_x = 0;
double mean_y = 0;
for(int i=0;i<var1.length;i++){
    mean_x += var1[i] / var1.length;
    mean_y += var2[i] / var1.length;
}
for(int i=0;i<var1.length;i++){
    double dlt_x = var1[i]-mean_x;
    double dlt_y = var2[i]-mean_y;
    tsq_x += (dlt_x * dlt_x);
    tsq_y += (dlt_y * dlt_y);
    tot_coproduct += (dlt_x * dlt_y);
}
res=tot_cp/(double)(Math.sqrt(tsq_x)*Math.sqrt(tsq_y));

where res, tsq_x, tsq_y and tot_cp stand for, respectively, $R$, $\text{tot}_sq_x$, $\text{tot}_sq_y$, $\text{tot}_\text{coproduct}$ in equations (15), (16), (17) and (18).

4.5. Exporting Adjusted Graph and Geodesic Distance into a SNA File Format

As explained in Chapter 3, two options are created in exporting adjusted graph into a SNA file regarding the format of the file. Main criteria in designation of the respective formats were, first whether or not they are supported by a wide range of SNA software programs, and second whether or not they can be
edited via a text editor in case such kind of an option may help users rearrange the data. Accordingly, as noted in Chapter 3, Ucinet’s Data Language (“.dl”) file format and Pajek’s Network (“.net”) file format were designated as the alternative output options in exporting adjusted graph into an external file. In exporting geodesic distance into an external file, the only option is designated to create a space separated values again in ASCII (“.txt”) format.

Since the spatial configuration concerned may already have an attribute table and the features involved in the layer covering the respective spatial configuration can easily be identified by an existing field, it is considered that an option for the labeling of column headings of SNA files according to an existing field may facilitate importing the results of analysis of the adjusted graph or geodesics in other software programs back to GIS. For this purpose, two options are created for labeling of column headings of the files created for the adjusted graph or geodesic distance; (1) employment of an existing field as the column headings or (2) employment of the internal index number of the features as the column headings.

For this purpose an array (`etiket`) is created in order to store the information about column headings. The script produced for labeling of column headings and creation of SNA (adjusted graph) or text (geodesics) is integrated into the main script as long as possible by making use of existing routines. In this respect, Script 11 shows Jython script created for defining header information and labeling of column headings in OpenJUMP;

Script 11. Jython script created for defining header information and labeling of column headings in OpenJUMP.

```python
# creating labels for matrix headings
etiket = []
for i in range(obj):
    etiket.append(str(i+1))

if UAF.getModel().isSelected():
    for i in range(obj):
        ss = fc.features[i]
```

40
value = ss.getAttribute(CAF.getSelectedIndex())
if (type(value).__name__ ==
'org.python.core.PyInteger'):
etiket[i] = str(value)
else:
    if (type(value).__name__ ==
'org.python.core.PyFloat'):
        feader = value.__int__()
etiket[i] = str(feader)
else:
etiket[i] = value

betiket = []
for i in range(obj):
betiket.append(etiket[i])

# defining header information for SNA and text file
if NT.getModel().isSelected():
    if NW.getSelectedIndex() == 0:
        edgelist = open(fname + '.dl','w')
edgelist.write("dl n="+str(obj))
edgelist.write('
')
edgelist.write("format = edgelist1")
edgelist.write('
')
edgelist.write("labels embedded")
edgelist.write('
')
edgelist.write("data:"))
edgelist.write('
')
else:
        edgelist = open(fname + '.net','w')
edgelist.write("*Vertices "+str(obj))
edgelist.write('
')
for n in range(obj):
edgelist.write(str(n+1)+" "+chr(34)+etiket[n]+chr(34))
betiket[n] = str(n+1)
edgelist.write('
')
edgelist.write("*Arcs")
edgelist.write('
')
if GM.getModel().isSelected():
geodesic = open(fname + '.txt', 'w')
for i in range(obj):
geodesic.write(" " + etiket[i])
geodesic.write('
')
After creating labels for matrix headings and defining header information for SNA and text file, in other parts of the main script adjusted graph data and geodesics data are written to the respective files created for this purpose. For example, if two features intersect, the adjacency information about them is written to SNA file by using the following sub-script included in the script produced for the construction of adjusted graph:

```python
if (ser == 1) and (m != n):
    if NT.getModel().isSelected():
        edgelist.write(betiket[n])
        edgelist.write(" ")
        edgelist.write(betiket[m])
        edgelist.write('
')
```

Script given below writes the geodesics to a text (".txt") file created for this purpose (the respective script is located in the section of the main script devoted to the calculation of geodesic distance):

```python
if GM.getModel().isSelected():
    for mas in range(obj):
        geodesic.write(etiket[mas])
    for kas in range(obj):
        geodesic.write(" ")
        geodesic.write(str(dst[mas*obj+kas]))
        geodesic.write('
')
```

The above examples are drawn from Jython SSA Plugin file created for OpenJUMP. As the procedure works in a similar fashion also in other SSA plugins, I will suffice to give above scripts here. Nevertheless, with some modifications and changes the respective scripts can easily be integrated into other scripting languages that can be used in order to develop plugins in other FOSS4GIS.

### 4.6. Design of Graphical User Interface

In the design of GUI, it is intended to provide user with a simple and functional interface (see Figure 9). Accordingly, all the standard space syntax parameters (connectivity, total depth,
mean depth, global integration, local depth, local integration and control values) are calculated without requiring any further specification from the user. The default value for the calculation of local parameters (local depth and local integration) is set to 3, albeit it (radius for local) can also be changed via GUI. Calculation of intelligibility value, production of external files (creation of the network file for the produced adjusted graph and the text file for geodesics) that can be further processed in SNA software programs and specification of matrix headings for the external files are designed to be optional. The respective options for the production of external files and matrix headings are designed in such a way that once the user chooses to produce an external file, he or she is also requested to specify how to name the matrix headings. A detailed explanation for the use of GUI is given in the extended version of the user’s guide presented in Chapter 6 of this book.

Figure 9. Graphical User Interface of SSA Plugin in Jython for OpenJUMP.

The scripts used to create GUI for SSA Plugin in Jython for OpenJUMP can be given as below:
Plugins and Scripts Developed to Conduct SSA in FOSS4GIS

Script 12. Jython script for GUI in OpenJUMP.

```python
f = JFrame("Space Syntax Analysis")
f.setBounds(100,100,210,325)
f.setResizable(0)

tf1 = TextField("3")

CAF = Choice()
CAF.add("available fields")
CAF.setEnabled(0)

IV = JCheckBox("Calculate intelligibility value.", 0)
NT = JCheckBox("Create network data:", 0)

NW = Choice()
NW.add("dl")
NW.add("net")
NW.setEnabled(0)

GM = JCheckBox("Create geodesic matrix.", 0)
SF = JCheckBox("Save file(s) in a different location.", 0)
SF.setEnabled(0)

ino = ButtonGroup()
UII = JRadioButton("Use internal index.", 0)
UAF = JRadioButton("Use a field.", 0)
UII.setEnabled(0)
UAF.setEnabled(0)
ino.add(UII)
ino.add(UAF)

radioPanel = JPanel()
radioPanel.setLayout(GridLayout(2, 1))
radioPanel.add(UII)
radioPanel.add(UAF)
radioPanel.setBorder(BorderFactory.createTitledBorder(BorderFactory.createEtchedBorder(), "Options for matrix headings"))
radioPanel.setPreferredSize(Dimension(190, 80))

ll = Label("")
ll.setPreferredSize(Dimension(215, 5))

tamam = JButton("OK")
iptal = JButton("Cancel")

f.add(Label("Radius for local:"))
f.add(tf1)
f.add(IV)
f.add(NT)
f.add(NW)
```
Conceptualization of SSA in GIS and Structure of SSA Scripts

```python
f.add(GM)
f.add(SF)
f.add(radioPanel)
f.add(Label("Choose a field:"))
f.add(CAF)
f.add(ll)
f.add(tamam)
f.add(iptal)
f.setLayout(FlowLayout(0))
f.setAlwaysOnTop(1)

# action when an output file option is selected
def dmpressed(x):
    dm=NT.getModel().isSelected() or GM.getModel().isSelected()
    SF.setEnabled(dm)
    UII.setEnabled(dm)
    UAF.setEnabled(dm)
    CAF.setEnabled(dm and UAF.getModel().isSelected())
    NW.setEnabled(NT.getModel().isSelected())

# action when cancel button is pressed
def ipressed(x):
    f.dispose()

# action when an option is chosen for matrix headings
def ufpressed(x):
    if UAF.getModel().isSelected():
        CAF.removeAll()
        CAF.add("available fields")
        for atin in range(fs.attributeCount-1):
            CAF.add(fs.getAttributeName(atin+1))
        else:
            CAF.setEnabled(0)

def spaceSyntax(event):
    NT.actionPerformed = dmpressed
    GM.actionPerformed = dmpressed
    UII.actionPerformed = ufpressed
    UAF.actionPerformed = ufpressed
    tamam.actionPerformed = SS
    iptal.actionPerformed = ipressed

    f.show()
```
5. Operationalization of SSA in FOSS4GIS

Operationalization of SSA in different FOSS4GIS requires a knowledge base for coding in different scripting languages. Depending upon FOSS4GIS, we may usually need to set up a development environment in order to create and compile the plugins. Nevertheless, there are also other opportunities in some FOSS4GIS, for example we may easily develop scripts and compile them in MapWindow GIS thanks to the scripting tool developed for this purpose. In a similar fashion, there are some console applications in many FOSS4GIS and we can use the respective applications in order to test and develop the scripts that will be used in a plugin.

After all if we develop our scripts in Python, what all we need is actually a text editor and we may easily integrate our plugin into FOSS4GIS if we carefully analyze existing ones and the samples provided to help us develop scripts. Except for OrbisGIS all the plugins produced for SSA are compatible with the latest version available for FOSS4GIS (Table 6). In the case of OrbisGIS, the script is only compatible with Version 2.2.

Table 6. Compatibility of SSA Plugin with FOSS4GIS.

<table>
<thead>
<tr>
<th>FOSS4GIS</th>
<th>Latest version with which SSA Plugin is compatible:</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenJUMP</td>
<td>Version 1.5.2 stable rev.2930</td>
</tr>
<tr>
<td>gvSIG</td>
<td>Version 1.11.0 final (Build 1305)</td>
</tr>
<tr>
<td>Thuban</td>
<td>Version 1.2.2</td>
</tr>
<tr>
<td>OpenEV</td>
<td>Version 1.8 (FWtools Version 2.2.0)</td>
</tr>
<tr>
<td>Quantum GIS</td>
<td>Version 1.7.4 – Wroclaw built</td>
</tr>
<tr>
<td>MapWindow GIS</td>
<td>Version 4.8.6 (Final release)</td>
</tr>
<tr>
<td>SAGA GIS</td>
<td>Version 2.0.8</td>
</tr>
<tr>
<td>OrbisGIS</td>
<td>Version 2.2 (Paris)</td>
</tr>
<tr>
<td>R Project</td>
<td>Version 2.14.0</td>
</tr>
</tbody>
</table>
Since SSA Plugin is composed of certain standard sections mentioned in Chapter 4, what developers basically need in a FOSS4GIS in order to create a script or plugin for the analysis of the vector layers is actually the procedure to iterate over the features involved in a vector layer. Thus, in this chapter the main consideration is devoted to the respective procedure along with the explanations for the establishment of the minimum software environment required to produce scripts in certain FOSS4GIS.

5.1. Creating Plugins for OpenJUMP

In OpenJUMP there are three different options to create plugins. The first option is the employment of a platform that is capable of compiling jar files. For this purpose, in this study Eclipse platform is used in order to both compile and develop the plugin that can be used in order to calculate basic space syntax parameters for line networks. The second option is to use BeanShell scripts as plugins. There are many advantages of this option. First of all, it is easier for beginners to employ BeanShell console in order to see how scripts work. Once the script is produced, it can be called as a plugin via “Customize → BeanTools” in OpenJUMP. Nevertheless, it takes much more time to complete the job assigned to a script in BeanShell environment than jar plugin. Except for this negative side concerning the speed of the script, we can do almost everything via BeanShell plugins without compiling a jar plugin. The third option for the production of plugins in OpenJUMP is the Jython console and plugin option. We can activate the Jython console in OpenJUMP via “Customize → Python Console and Tools...”. Although compared with jar plugins Jython plugins are again slow, they work faster than BeanShell plugins.

In the subsequent part of this section, reader will be informed about the steps and environment required in each option available for the production of plugins in OpenJUMP.
5.1.1. Creating SSA Plugin in Java for OpenJUMP

In order to create Java plugins, we should first install a development environment (IDE – Integrated Development Environment) for the respective task (see How to make your plugin in ECLIPSE for a developer’s guide for OpenJUMP). Many OpenJUMP programmers prefer to use Eclipse IDE. We may also develop plugins using Netbeans (particularly for the development of GUIs) and JEdit. All these development environments (Eclipse, Netbeans and JEdit) are open source. In this study, OpenJUMP Java SSA Plugin is developed by using Eclipse IDE. Thus, in order to follow the procedure described here for Java SSA Plugin, we should first download “Eclipse IDE for Java Developers” (Version: Indigo Service Release 2) (eclipse-java-indigo-SR2-win32.zip) from the web page of the Eclipse.

After downloading the software, we should extract it into a folder in which we can create a short cut for “eclipse.exe” and later move the short cut to our desktop or an application folder in order to start the program. Java source files for OpenJUMP SSA Plugin should also be downloaded from the web page of SSA Plugin (http://mekandizim.mersin.edu.tr/) and extracted into a folder. Once we start eclipse it will ask to open a workspace where our projects will be stored. We can choose the default place offered to us and initiate the program. After starting the program, first of all, we can easily import “Existing Projects into Workspace” (via “File → Import → General → Existing Project into Workspace”) by choosing the source file folder (via “Select root directory”) created after extracting the compressed folder available at the web page of the Plugin (Figure 10).

In this respect, when we are prompted to select the root directory, we should navigate to the folder including “MekanDizim.Araclarim” folder. When we select the respective root folder and press “OK”, MekanDizim.Araclarim can be chosen as a workspace (Figure 11). Alternatively, we can build a plugin for OpenJUMP from scratch. For this purpose we should be familiar with SourceForge Project page for the JUMP Pilot project.
Figure 10. Importing an existing project into Eclipse.
Plugins and Scripts Developed to Conduct SSA in FOSS4GIS

Figure 11. Eclipse IDE after opening the existing workspace.
In order to create a jar plugin in Eclipse, we should first create a new project in Eclipse using “File → New → Project → Java → Java Project”. After defining the name of the project (“Project name”) as “MekanDizim.Araclarim”, we can press “Finish” in order to create new project file. After creating MekanDizim.Araclarim project, we should also include libraries that will be used in the scripts. For this purpose, we need an installed OpenJUMP. By clicking the right mouse button while the mouse pointer is on the name of the project, we can select the project properties and then by selecting “Java Built Path → Libraries” we can add external libraries (JARs) that will be needed by our OpenJUMP plugin (Figure 12 and Figure 13).

After creating an empty project and adding the external jars to the project, we can begin to create our plugin functionality. For this purpose, we should create a package and add a Class to the respective package via right clicking the name of the project (project folder) and selecting “New → Class” (Figure 14). In the coming dialog box (see Figure 15), define “Package:” as “spacesyntax.tools”, “Name:” as “MekanDizimPlugIn” and define other options as shown in the figure. We should be aware of the following naming rules (see How to make your plugin in ECLIPSE):

- for JUMP plugins the class name must end with PlugIn
- Java Classes should start with uppercase letters
- further we should create our class in a package or name space as we did using spacesyntax.tools

After completing above procedure, our MekanDizimPlugIn class has to be made as a subclass of AbstractPlugIn by adding extends AbstractPlugIn. In OpenJUMP in general two PlugIn methods (method initialize() and method run()) are required to interact (see Script 13):

1. method initialize() is required to register and to create a menu item;
2. method run() contains the script which is run if the menu entry is activated.
Figure 12. Selecting “Properties” in order to add external JARs into a project in Eclipse.
Figure 13. Adding external JARs into a project in Eclipse.
Plugins and Scripts Developed to Conduct SSA in FOSS4GIS

Script 13. Summarized content of MekanDizimPlugIn.java.

```java
public class MekanDizimPlugIn extends AbstractPlugIn{
    private MekanDizimEngine engine = new MekanDizimEngine();
    public MekanDizimPlugIn()
    {
        // empty constructor
    }
    public void initialize(PlugInContext context)
    throws Exception {
        FeatureInstaller featureInstaller = new FeatureInstaller(context.getWorkbenchContext().
        );
        featureInstaller.addMainMenuItem( this,
            new String[] {"Graph Theoretic"}, //menu path
            "Space Syntax Analysis", //name
            false, //checkbox
            null, //icon
            new MultiEnableCheck().add(context.getCheckFactory().createTaskWindowMustBeActiveCheck()).
                add(context.getCheckFactory().createAtLeastNLayersMustExistCheck(1)); //enable check
    }
    public static MultiEnableCheck createEnableCheck(WorkbenchContext workbenchContext) {
        EnableCheckFactory checkFactory = new EnableCheckFactory(workbenchContext);
        return new MultiEnableCheck()
            .add(checkFactory.createWindowWithNamePanelMustBeActiveCheck());
    }
    public boolean execute(PlugInContext context)
    throws Exception {
        // code to be run
    }
}
```
Figure 14. Creating a package and Class file under a project in Eclipse.
We also need to create the following class in order to load the PlugIn into OpenJUMP. The respective class has to be called “Extension” in the end of class name in order to make it possible use the PlugIn in OpenJUMP (MekanDizimExtension.java);

```java
package spacesyntax.tools;
import com.vividsolutions.jump.workbench.plugin.Extension;
import com.vividsolutions.jump.workbench.plugin.PlugInContext;

public class MekanDizimExtension extends Extension{
    public void configure(PlugInContext context) throws Exception{
        new MekanDizimPlugIn().initialize(context);
    }
}
```

Figure 15. Creating a new Java Class for a project in Eclipse.
After creating the classes explained above and with the required characteristics, we can start to write our code inside, “public boolean execute(PlugInContext context) throws Exception { code to run }”

In the particular case of SSA Plugin, when the menu item is selected, the script produced for GUI is run and it is connected to another Class (MekanDizimEngine.java) responsible for the construction of the adjusted graph, calculation of the geodesics and space syntax parameters together with intelligibility value.

5.1.2. BeanShell Console and Plugin in OpenJUMP

We can execute scripts written in Java by using BeanShell Console (Figure 16) in OpenJUMP (“Customize → BeanShell Console...”). BeanShell invented by Patrick Niemeyer is actually a lightweight scripting tool for Java and we can explore and test specific methods and functions by making use of it (see Beanshell, Simple Java Scripting, version 1.3). Similar kinds of BeanShell consoles are also available in OrbisGIS and Kosmo GIS. Since Kosmo GIS makes use of OpenJUMP libraries, BeanShell scripts produced for OpenJUMP can easily be run in Kosmo without making any major changes.

Figure 16. BeanShell Console in OpenJUMP.

Alternatively “Scripting → BeanShellEditorPlugin” (Figure 17) developed by Michaël Michaud (2009) can also be used in OpenJUMP (we should include the respective plugin file (bsheditor4jump-0.2.jar) under ext folder) in order to produce and run BeanShell scripts. There is a detailed user’s guide written by Michaud (2004) for the respective plugin.
One of the most important parts of the script produced for SSA Plugin is actually the part devoted to the iteration of vector features involved in a spatial configuration and checking the intersection between them. Once we iterate over the vector features and test intersection, we can easily construct adjusted graphs. Figure 18 shows the scripts produced for the construction of adjusted graph and calculation of geodesics by using BeanShell Console in OpenJUMP and iterating over the features covered in a vector layer. In the respective script, first we need to define the layer in which the features will be iterated in order to test whether or not they intersect with each other. In the next step feature collection ($fc$) is defined and subsequently variables that will be used in order to construct adjusted graph are defined according to the properties of $fc$. Finally, each feature involved in $fc$ is iterated by using a for – next loop available for this purpose in OpenJUMP (for (f : fc.features){}). In the respective loop each feature (f) is read sequentially and checked whether or not they intersect other features by using the relevant information and functions (getGeometry and intersects) available for this purpose.
Figure 18. Construction of adjusted graph and calculation of geodesics by using BeanShell Console in OpenJUMP.
5.1.3. Jython Console and Plugin in OpenJUMP

Another option in OpenJUMP for running and testing scripts is Jython Console that can be activated via “Customize → Python Console and Tools...” (activate Jython menu-tool) and then “Tools → Show Console”.

![Jython Console in OpenJUMP](image)

Figure 19. Jython Console in OpenJUMP.

A similar kind of Jython console is also available in gvSIG. Jython is actually an implementation of Python programming language written in Java. In Jython we can import and use any Java class. As it is also the case for SSA Plugin, for example, a user interface in Jython could be written with Swing or AWT. Thus, with language specific modifications a script produced for BeanShell can easily be run in Jython console.

In the construction of adjusted graph, the same set of built-in functions and methods (getGeometry and intersects) are used in Jython in order to iterate over the features covered by a vector layer (see Figure 20). Developer should notice that 2 dimensional arrays in BeanShell have been translated into 1 dimensional form in Jython by following the required procedure (for example \texttt{sna(n*obj+m)}). Overall, Jython scripts work faster than BeanShell scripts (see also http://www.javaworld.com/javaworld/jw-04-2002/jw-0405-scripts.html?page=3).
Figure 20. Construction of adjusted graph and calculation of geodesics by using Jython Console in OpenJUMP.
5.2. Creating Plugins for gvSIG

There are again different options to produce plugins in gvSIG. The first option similar to the one available in OpenJUMP is the employment of a Java IDE and compiler. In this book, this option is not used for the production of space syntax plugins for gvSIG. The second option which is available in development version of gvSIG (version 2.0) is the employment of BeanShell or Groovy console in order to run and test Java scripts. The last one is the production of plugins by using Jython (“File → Scripting → Jython console”). Similar to OpenJUMP, users can test and explore the way how scripts works in gvSIG via Jython console (Figure 21). In this study, space syntax plugin for gvSIG is produced by employing last option. The respective plugins can be run via both menu bar and toolbar.

Figure 21. Jython Console in gvSIG.

Plugins created for gvSIG is placed under a folder named “extensiones”. In gvSIG we should first create a folder for our plugin project (in the case of SSA Plugin since there are two options for GUIs, two separate folders (SSA and Space Syntax) are created for this purpose). The next step is the creation of a config.xml file having content similar to the following one in our plugin folder:

---

7 Groovy is a dynamic language for the Java Virtual Machine and it has additional power and features inspired by languages like Python, Ruby and Smalltalk (see http://groovy.codehaus.org/).
As it is explained in a much more detailed context in *Scripting guide Version 2* for gvSIG 1.0, we should modify config.xml in order to create our own plugin. For example “menu text=” tag defines “where and how the entry is added to the menu bar”. In order to create a new entry in the “File → Scripting” menu with the name “Space Syntax Analysis”, "Archivo/Scripting/Space Syntax Analysis" should be entered. We should also define the properties (such as “tooltip=” for the text to be shown when the cursor is on the menu entry, “action-command=” for the actions to be performed when the user click menu entry, “icon=” for the icon to be displayed) for the respective tag. In a similar fashion we can also create a tool bar button for our plugin by modifying “tool-bar name=” tag.

There are two options for the production of GUIs in gvSIG ) (Figure 22): (1) employment of Jython scripts (user interface produced by employing Swing or AWT libraries) and (2) xml
dialogs. If we prefer to use xml dialog that is easier to build by using ThinG created by Dirk Moebius (2004), we should indicate this in “action-command=’” reserved for this purpose. In config.xml “action-command=’” tag can actually initiate two types of actions; (1) displaying an xml dialog window as a GUI having connection to a Jython script, and (2) directly running a script. As it is explained in Scripting guide Version 2, both options take several parameters (see for example following examples for SSA Plugin):

1. action-command = "run(fileName='gvSIG/extensiones/SSA/spacesyntax.py',language='jython')"
2. action-command = "show(fileName='gvSIG/extensiones/SpaceSyntax/spacesyntax.xml', language='jython', title='Space Syntax Analysis', width=252, height=233)"

gvSIG Jython GUI

gvSIG xml GUI

Figure 22. GUI alternatives in gvSIG.

Compared with other consoles, Jython Console of gvSIG is a well developed one in which there is an option to automatically complete the script written down by choosing among the options offered (see Figure 23).
Operationalization of SSA in FOSS4GIS

Figure 23. Auto-complete feature of Jython Console in gvSIG.

Figure 24 shows Jython script that can be used in order to construct adjusted graph by iterating over the features covered by a vector layer in gvSIG. Whether or not features intersect with each other can easily be tested by first converting geometry of each feature to JTS geometry (toJTSGeometry) and then checking the intersection between them by using intersects function. After constructing the adjusted graph, geodesics between the features can easily be calculated by employing the script given in Figure 25 in gvSIG.

Figure 24. Construction of adjusted graph by using Jython Console in gvSIG.
Figure 25. Calculation of geodesics by using Jython Console in gvSIG.

5.2.1. Creating GUI using xml

xml dialog windows can easily be developed by employing Thinlet which is a GUI toolkit graphic library in Java. As it is remarked above, there is an open source software program called ThinG in order to design GUIs in xml format (Figure 26). As a Thinlet GUI editor, ThinG can be used to design our GUI for gvSIG Plugin. ThinG is a actually a fast and flexible alternative to Swing. Content of xml dialog that is partly created by employing ThinG and used in SSA Plugin is given below;

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- generated by ThinG, the Thinlet GUI editor -->
<panel columns="2" gap="4" left="5" top="5">
<separator colspan="2" height="10" width="1"/>
<separator colspan="2" height="10" width="1"/>
<script language="jython" method="init" src="spacesyntax.py"/>
<label colspan="1" halign="left" text="Radius for local:"/>
<textfield name="localrad" text="3"/>
<checkbox colspan="2" name="IV" text="Calculate intelligibility value." tooltip="option to calculate intelligibility value."/>
<checkbox colspan="1" name="NT" text="Create network data:" tooltip="option to create network file." action="clickDL(thinlet)"/>
...  
```
Figure 26. ThinG for creating xml GUIs.
5.2.2. Creating GUI using Jython

Alternatively we can create GUIs for our SSA Plugin using Swing or AWT libraries. Jython console can be used to test the scripts produced for this purpose. Below is the script produced to create GUI for SSA Plugin in gvSIG by using Swing and AWT libraries:

Script 14. Jython script for GUI in gvSIG.

```python
f = JFrame("Space Syntax Analysis")
f.setBounds(100,100,210,325)
f.setResizable(0)

tf1 = TextField("3")
CAF = Choice()
CAF.add("Choose a field")
CAF.setPreferredSize(Dimension(96, 20))
CAF.setEnabled(0)

IV = JCheckBox("Calculate intelligibility value.", 0)
NT = JCheckBox("Create network data:", 0)
NW = Choice()
NW.add("dl")
NW.add("net")
NW.setEnabled(0)

GM = JCheckBox("Create geodesic matrix.", 0)
SF = JCheckBox("Save file(s) in a different location.", 0)
SF.setEnabled(0)

ino = ButtonGroup()
UII = JRadioButton("Use internal index.", 0)
UAF = JRadioButton("Use a field.", 0)
UII.setEnabled(0)
UAF.setEnabled(0)
ino.add(UII)
ino.add(UAF)

radioPanel = JPanel()
radioPanel.setLayout(GridLayout(2, 1))
radioPanel.add(UII)
radioPanel.add(UAF)
radioPanel.setBorder(BorderFactory.createTitledBorder("
radioPanel.setPreferredSize(Dimension(190, 80))

ll = Label(""
ll.setPreferredSize(Dimension(215, 5))
```
tamam = JButton("OK")
iptal = JButton("Cancel")

f.add(Label("Radius for local:"))
f.add(tf1)
f.add(IV)
f.add(NT)
f.add(NW)
f.add(GM)
f.add(SF)
f.add(radioPanel)
f.add(Label("Choose a field:"))
f.add(CAF)
f.add(ll)
f.add(tamam)
f.add(iptal)
f.setLayout(FlowLayout(0))
f.setAlwaysOnTop(1)
f.show()

# action when an output file option is selected
class disaver(java.awt.event.ActionListener):
def actionPerformed(self, event):
    dm = NT.getModel().isSelected() or
    GM.getModel().isSelected()
    SF.setEnabled(dm)
    UII.setEnabled(dm)
    UAF.setEnabled(dm)
    CAF.setEnabled(dm and UAF.getModel().isSelected())
    NW.setEnabled(NT.getModel().isSelected())
    NT.addActionListener(disaver())
    GM.addActionListener(disaver())

class ufim(java.awt.event.ActionListener):
def actionPerformed(self, event):
    CAF.setEnabled(UAF.getModel().isSelected())
    UAF.addActionListener(ufim())
    UII.addActionListener(ufim())

# action when Cancel button is pressed
class kapayici(java.awt.event.ActionListener):
def actionPerformed(self, event):
    iptal.addActionListener(kapayici())

# action when OK button is pressed
class baslatici(java.awt.event.ActionListener):
def actionPerformed(self, event):
    main()
    f.dispose()
tamam.addActionListener(baslatici())
5.3. Creating Scripts for OrbisGIS

There are basically two options to create scripts that can be executed in OrbisGIS. Similar to the options available both in OpenJUMP and gvSIG, in OrbisGIS one can develop plugins using a Java IDE and compiler such as Eclipse platform. The second option is the employment of Jython console (Figure 27) and execution of produced scripts via the respective console (“View → Beansheil Console”). In this study, space syntax plugin for OrbisGIS is produced by employing the second option. As it is remarked at the beginning of this chapter, due to the unresolved issues encountered by the author in the recent version of the software program, SSA BeanShell script produced for OrbisGIS can only be run in Version 2.2 (Paris).

![Figure 27. BeanShell Console in OrbisGIS.](image)

If a vector layer is selected in OrbisGIS, BeanShell script shown in Figure 28 constructs the adjusted graph (sna) and, subsequently, calculates the geodesic distance for the spatial configuration concerned. In the respective script, features covered by a vector layer in OrbisGIS is iterated by using the same set of built-in functions and methods (getGeometry and intersects) that are also used in OpenJUMP thanks to JTS. Parallel to OpenJUMP and gvSIG, in order to create GUI in OrbisGIS for SSA Plugin, Swing and AWT libraries are used. Figure 29 shows the scripts used for this purpose in OrbisGIS. The respective script is also typical for those FOSS4GIS having BeanShell (or Jython) Console.
Figure 28. Construction of adjusted graph and calculation of geodesics by using BeanShell Console in OrbisGIS.
Plugins and Scripts Developed to Conduct SSA in FOSS4GIS

Figure 29. Creating GUI of SSA Plugin by using BeanShell Console in OrbiGIS.
5.4. Creating Plugins for Quantum GIS

The easiest way to create a plugin in Quantum GIS is the employment of several tools together such as “Plugin Builder” plugin, QT Creator IDE and tools (see particularly Kepoğlu and Düzgün (2008), and Kepoğlu (2011) for the development of plugins in QGIS). “Plugin Builder” plugin is created by Gary Sherman (2012) in order to create a plugin template. The respective plugin (Figure 30) can be run via “Plugins → Plugin Builder... → Plugin Builder...” after installing it via “Plugins → Fetch Python Plugins...”. We can also use existing plugins in order to develop our plugin.

![Figure 30. GUI of QGIS Plugin Builder.](image)

There is an online tutorial for creating plugins in QGIS (see Tutorial – Building A Simple Plugin). After creating a template plugin by using “Plugin Builder”, the content of the template files should be modified in order to build out our plugin. For example, we can develop a GUI for our plugin by using QT Creator (see Figure 31) and save it as “.ui” file. The respective file should be compiled by using pyuic4. In a similar fashion, if we use an icon for our plugin it can be compiled by using pyrcc4. After creating an icon by using, for example, GIMP (GNU Image Manipulation Program) in “.png” format, we should modify the content of resources.qrc file containing information about the respective icon. The content of resources.qrc is given below:
Figure 31. QT Creator for creating Python GUIs.
Content of resources.qrc;

```xml
<RCC>
    <qresource prefix="/plugins/spacesyntax">
        <file>icon.png</file>
    </qresource>
</RCC>
```

GUI (“.ui” file) and resources (“.png” and `resources.qrc`) can be compiled by using following commands;

```bash
pyuic4 -o ui_qgsspacesyntaxbase.py iletisim.ui
pyrcc4 -o resources.py resources.qrc
```

where `ui_qgsspacesyntaxbase.py` and `resources.py` are output files covering scripts for GUI and icon resource in Python.

As it is seen above, we can easily design our plugin icon and GUI by using GIMP and Qt Creator, and later compile them in order to use in our plugin (although `pyrcc4` can be found under “C:\Program Files\Quantum GIS\bin\” folder or “C:\OS Geo4W\bin\”, it would be better to download Qt SDK version 1.2.1 rather only Qt Creator in order to prevent any compilation error). After building a working plugin template for our plugin, we can produce the scripts required to conduct space syntax analysis. In this respect, Python Console (Figure 32) available in QGIS can be particularly used in order to develop scripts by running and testing them (“Plugins → Python Console”).

![Python Console](image)

**Figure 32.** Python Console in Quantum GIS.
Figure 33 shows how features covered by a vector layer are iterated in Quantum GIS with particular reference to the part of SSA Plugin responsible for the construction of adjusted graph (featureAtId(id, feature), geometry() and intersects()). Subsequent figure illustrates the calculation of geodesics in QGIS.

```python
>>> provider = layer.dataProvider()
>>> feat = QgsFeature()
>>> beat = QgsFeature()
>>> sna = []
>>> obj = layer.featureCount()
>>> for i in range(obj):
...     sna.append(1)
...     for j in range(obj):
...         sna[i].append(0)
...     for s1 in range(obj):
...         provider.featureAtId(s1, feat)
...         for s2 in range(obj):
...             provider.featureAtId(s2, beat)
...             kesisme = feat.geometry().intersects(beat.geometry())
...             if kesisme and (s1 != s2):
...                 sna[s1][s2] = 1
...```

Figure 33. Construction of adjusted graph by using Python Console in Quantum GIS.

```python
>>> dst = sna
>>> for mas in range(obj):
...     for kas in range(obj-1):
...         fas = 1
...         for sas in range(obj):
...             if ((dst[mas][sas] == (kas+1)) and (sas != mas)):
...                 for tas in range(obj):
...                     if ((dst[sas][tas] == 1) and (tas != mas)) and
...                         ((dst[mas][tas] > (kas+1)) or (dst[mas][tas] == 0)):
...                         dst[mas][tas] = kas + 2
...                         fas = fas + 1
...                     if (fas == 1): break
...     ...
```

Figure 34. Calculation of geodesics by using Python Console in Quantum GIS.
5.5. Creating Plugins for Thuban

Although Thuban is a seemingly simple FOSS4GIS, like many other FOSS4GIS it has also been designed to be extensible. Information about how to develop plugins (extensions) in Thuban can be found in Coles, Wagner and Koormann (2004). When Thuban starts, it imports a file named thubanstart.py in which we can include statements to import our plugins into the program. Within this context, in the case of SSA Plugin it is necessary to add the following line to the thubanstart.py in order to make it possible for Thuban to start with the opportunity to conduct SSA.

```python
import spacesyntax.py
```

SSA code is stored in spacesyntax.py file. Analysis of existing extensions (plugins) located under “Thuban\Extensions” may help us develop our own plugin. In order to develop the scripts used in SSA Plugin, a similar kind of strategy is employed. All scripts should be arranged in connection with the Thuban context. In the case of SSA Plugin, this is arranged via;

```python
def SpaceSyntax(context):
```

In order to connect a plugin function to the menu, it has to be registered to the Thuban framework. For this purpose, following script should be included, for example, in SSA Plugin (spacesyntax.py);

```python
# register the new command
registry.Add(Command('SpaceSyntax',
    _('Space Syntax Analysis'), SpaceSyntax,
    helptext = _('Conduct Space Syntax Analysis'),
    sensitive = _has_selected_shape_layer))
```

The last statement makes the new command unselectable if there is no selected layer, which prevents the irrelevant call of the plugin. Following statements arrange the menu for the plugin and add it as a new entry to the menu;
Plugins and Scripts Developed to Conduct SSA in FOSS4GIS

# find extension menu (create a new one if not found)
extensions_menu = main_menu.FindOrInsertMenu('spacesyntax',
    ('Space Syntax Analysis'))

# finally add the new entry to the menu
extensions_menu.AddItem('SpaceSyntax')

In order to iterate over the features covered by a vector layer in Thuban, we can use the script given in Script 15. Nevertheless, since there is no built function in Thuban in order to check the intersection between the features involved in a vector layer, whether or not features intersect each other should be checked by using bounding box function and iterating over the segments placed along each feature as it is discussed in Chapter 4 (Section 4.1.3. and Section 4.1.4.).

Script 15. Python script created for iterating over the features covered by a vector layer in Thuban.

    self.layer = layer
    obj = self.layer.NumShapes()

    for s1 in self.layer.ShapeStore().AllShapes():
        i1 = s1.ShapeID()
        k1 = s1.Points()
        l1 = k1[0]
        row1 = len(l1)
        bbox = self.layer.ShapesBoundingBox([i1])

        for s2 in self.layer.ShapeStore().AllShapes():
            i2 = s2.ShapeID()
            k2 = s2.Points()
            l2 = k2[0]
            row2 = len(l2)
            sbox = self.layer.ShapesBoundingBox([i2])

where s1 and s2 stores the features that will be checked for intersection, l1 and l2 stores the array of coordinates for the first part of the respective features. As it is discussed in Chapter 4, the algorithm used in some SSA Plugins created for FOSS4GIS having no built-in function for checking the intersection between features involved in a layer assumes that features are not multi-line features. If there is a multi-line feature, it only takes into account the first part of the respective feature without iterating over the other parts involved in a geographic feature. That’s why
in Script 15 although all the coordinates of the features \((k1 \text{ and } k2)\) are read, only the coordinates of the points placed along the first part of each feature \((11 \text{ and } 12)\) are used in the calculation of intersection between features.

5.6. Creating Plugins for OpenEV

OpenEV is another relatively simple desktop FOSS4GIS for which it is again possible to extend the functionality of the program by creating plugins using Python. In contrast to Thuban for which there is no console application in order to run scripts, there is an opportunity in OpenEV to test and run the scripts via Python Shell (“Edit → Python Shell...”) which facilitates the learning process for a user (Figure 35).

Figure 35. Python Console in OpenEV.

If we have a polyline layer loaded into OpenEV, we can run the scripts shown in Figure 36 in order to read the number of features \((\text{obj})\) involved in the layer, the number of nodes \((\text{row1})\) involved in the first polyline feature and \(x, y\) coordinates of the first node of the respective feature. In contrast to Thuban, in OpenEV for which there is also no built-in function for checking the intersection between features, multi-line features can be taken into account without iterating over parts.
Figure 36. Iterating over geographic features in a vector layer by using Python Console in OpenEV.

Since there is no built-in function in OpenEV in order to detect the intersection between the features involved in a vector layer, as it is mentioned in Chapter 4 we need to produce the required scripts for this purpose. There is an online tutorial about how to create a new plug-in OpenEV (see Extending OpenEV - Tools and Commands and OpenEV Customization). Following scripts shows how a menu entry for SSA Plugin is added to OpenEV’s menu bar and how to store a handle to run GUI.

Script 16. Python script created for defining the menu entry and the handle for SSA Plugin in OpenEV.

```python
class SpaceSyntax(gviewapp.Tool_GViewApp):
    def __init__(self, app=None):
        gviewapp.Tool_GViewApp.__init__(self, app)
        self.init_menu()

    def launch_dialog(self, *args):
        self.win = SSDialog()
        self.win.show()
```

--- Interactive Python Interpreter ---

```python
>>> layer = gview.app.sel_manager.get_active_layer()
>>> shapes = layer.get_parent()
>>> obj = len(shapes)
>>> si = shapes[0]
>>> rsi = si.get_nodes()
>>> ei = si.get_nodes(0)
>>> rowi
(492750.53125, 4423244.5, 0.0)
>>> x = ei[0]
>>> y = ei[1]
>>> x
492750.53125
>>> y
4423244.5

```
def init_menu(self):
    self.menu_entries.set_entry("Tools/Space Syntax Analysis", 2, self.launch_dialog)

TOOL_LIST = ['SpaceSyntax']

5.7. Creating Plugins for MapWindow GIS

In MapWindow there is an excellent tool to test, run and compile the scripts written in both VB.Net and C# (Figure 37). We can activate the respective tool via “Plug-in → Scripts”. There are sample scripts for both VB.Net and C#. We can run them and modify the existing scripts to learn how the scripts work. Once we finish composing a plugin and testing it, we can compile it as a “.dll” plugin again by using the respective tool to be loaded each time when MapWindow GIS starts.

Figure 37. Scripting tool in MapWindow GIS.
Plugins and Scripts Developed to Conduct SSA in FOSS4GIS

By using plugin template or existing source files for different plugins, we can easily create plugins in MapWindow. There is also some user and developer guides for developing plugins in MapWindow GIS (see for example Watry, Ames and Michaelis (2007) and Marchionni (2008)). Nevertheless, the employment of plugin template provided at the old wiki is more instructive in many respects (see MapWindow Developer’s Guide and Plugin Development with Visual C-sharp).

Although there is a built-in function in MapWindow GIS in order to check the intersection between the geographic features, the respective function seems to fail in checking the intersection between the respective features (for the details see http://www.mapwindow.org/phorum/read.php?2,23745). Hence, SSA Plugin for MapWindow does not make use of the available built-in function in MapWindow for checking the intersection between the geographic features.

The script in Figure 38 shows how to iterate through the features involved in a vector layer in order to check the intersection between the bounding boxes (BBs) of the respective features for the construction adjusted graph in MapWindow GIS by using VB.Net. If we run the respective script, the output will be displayed in a message box showing the adjacency between the features according to the intersection between BBs.

Checking intersection between BBs of the features is performed at the following part of the script shown in Figure 38;

```vbnet
If (xmax < umin Or ymin > vmax) Then
    bonuc = "0"
ElseIf (xmin > umax Or ymin > vmax) Then
    bonuc = "0"
ElseIf (xmax < umin Or ymax < vmin) Then
    bonuc = "0"
ElseIf (xmin > umax Or ymax < vmin) Then
    bonuc = "0"
Else
    bonuc = "1"
End If
```
Figure 38. VB.Net script produced for iteration of features in MapWindow GIS in order to check intersection between BBs.
Plugins and Scripts Developed to Conduct SSA in FOSS4GIS

5.8. Creating Plugins for SAGA

In order to create plugins for SAGA GIS in Windows, we are required to install Microsoft Visual C++ 2010 as the development platform (see Figure 39). We can also compile source files in Linux. There is a well written guide for programming SAGA modules by Victor Olaya (2004: 185-194). SSA Plugin is also developed for SAGA by using the respective guide and following some trial-error processes (analyzing the other plugins for finding some clues about the functions or procedures required in the plugin).

Although SAGA has been developed by using Microsoft Visual C++, unlike SAGA, it is not free software. Yet, we can download the compiler for MS Visual C++ instead of the development environment. As it is the case for other versions of SSA Plugin, it is assumed that the reader has some knowledge about programming and preferably in Visual C++ for the particular case of SAGA. After installing Microsoft Visual C++ 2010 and downloading the source files for SAGA, we will notice that in SAGA source files there is a folder for the creation of plugins (“src\modules_template”). By modifying the files in the respective template folder, we can easily produce our own plugins. Examination of source files of other modules may also help us develop our plugin. Nevertheless, in SAGA there is no console application through which we can test our scripts line by line as it is the case with OpenJUMP, gvSIG, QGIS, OpenEV, MapWindow and OrbisGIS.

Below is the list of files located under “src\modules_template”;

- Template.vcproj
- Template.h
- Template.cpp
- MLB_Interface.h
- MLB_Interface.cpp

We can begin to create SSA Plugin (namely module in SAGA) by renaming the files and changing the content of them according to this renaming process. Accordingly, following list of files has been created in the case of SSA Plugin;
Figure 39. Microsoft Visual C++ for developing and compiling plugins for SAGA GIS.
Plugins and Scripts Developed to Conduct SSA in FOSS4GIS

- **space Syntax.vcproj**  
- **space Syntax.h**  
- **space Syntax.cpp**  
- **MLB_Interface.h**  
- **MLB_Interface.cpp**

**space Syntax.vcproj** should be opened by using a text editor such as notepad and all Template entries with extension “.h” and “.cpp” should be replaced with space Syntax. All MLB_Interface entries should be kept as they are without any renaming. After finishing replacements and saving the project file, it can be opened with Visual C++.

**MLB_Interface.cpp** defines the structure of a module (see the script given below);

**Script 17. Content of MLB_Interface.cpp for SSA Plugin.**

```cpp
// 1. Include the appropriate SAGA-API header...
#include " MLB_Interface.h"

// 2. Place general module library information here...
const SG_Char * Get_Info(int i)
{
    switch( i )
    {
    case MLB_INFO_Name: default:
        return( _TL("Space Syntax Analysis Module") );
    case MLB_INFO_Author:
        return( SG_T("Burak Beyhan (c) 2011") );
    case MLB_INFO_Description:
        return( _TL("This module calculates basic space syntax parameters for line networks") );
    case MLB_INFO_Version:
        return( SG_T("1.0") );
    case MLB_INFO_Menu_Path:
        return( _TL("Shapes|Graph Theoretic") );
    }
}

// 3. Include the headers of your modules here...
#include "space Syntax.h"

// 4. Allow your modules to be created here...
CSG_Module * Create_Module(int i)
```

86
Information about new module is defined in MLB_Interface.cpp. For example the part of the script given below is used in order to define menu path via which module can be run:

```
case MLB_INFO_Menu_Path:
    return( _TL("Shapes|Graph Theoretic ") );
```

where “Shapes|Graph Theoretic” means that module will be available via “Modules → Shapes → Graph Theoretic → Space Syntax Analysis”. Alternatively, we can also define menu path for the plugin in space_syntax.h file as given below;

```c
#include "MLB_Interface.h"

class Cspace_syntax : public CSG_Module
{
public:
    Cspace_syntax(void);
    virtual const SG_Char * Get_MenuPath (void)
    {return( _TL("A:Shapes") ); }

protected:
    virtual bool On_Execute (void);
}
```

We can define and develop as many modules as we want in MLB_Interface.cpp by including the headers of the respective modules inside it. Under “Graph Theoretic” heading currently we have only one module created for SSA (header file for the respective module is space_syntax.h). In MLB_Interface.cpp, each module file is defined as a C++ class. For this purpose, each module has to be specified as a class inside the Create Module method (Olaya, 2004: 187):
Plugins and Scripts Developed to Conduct SSA in FOSS4GIS

```c
CSG_Module * Create_Module(int i)
{
    // Don't forget to continuously enumerate the case
    // switches when adding new modules! Also bear in
    // mind that the enumeration always has to start with
    // [case 0:] and that [default:] must return NULL!
    switch( i )
    {
    case 0:    return( new Cspace_syntax );
    default:   return( NULL );
    }
}
```

After defining the main structure of the library as described above, SSA module can be developed by producing the scripts for different parts of the plugin discussed in Chapter 4. Actually template module which was renamed as space_syntax.cpp involves the important features of SAGA modules together with explanations and comments about how to create a module. Since SAGA modules are open source, we can also analyze and use them in order to develop our own module.

As part of plugin (module), we should first construct a GUI in order to retrieve information from the user for the execution of the codes produced for SSA. This part of module can be called as the constructor used in order to specify parameters via a GUI. The code used as the constructor for SAGA SSA Plugin is given below;

**Script 18. Constructor part of space_syntax.cpp for SSA Plugin.**

```c
Cspace_syntax::Cspace_syntax(void){
    CSG_Parameter  *pNode;
    Set_Name        (_TL("Space Syntax Analysis"));
    Set_Author      (SG_T("Burak Beyhan (c) 2012"));
    Set_Description (_TW(
                 "This module performs space syntax analysis.\n                 "Calculates basic space syntax parameters."
    ));

    // script for GUI
    Parameters.Add_FilePath(
        NULL,  "FILENAME",  _TL("File"),
        _TL(""),
        CSG_String::Format(SG_T("%s|\%s"),
            _TL("All Files"),  SG_T("*.*)")
    );
}```
Operationalization of SSA in FOSS4GIS

    }, NULL, true
    
    pNode = Parameters.Add_Shapes(
        NULL, "LINES", _TL("Lines"),
        _TL(""),
        PARAMETER_INPUT
    );
    
    Parameters.Add_Value(
        NULL, "LIT", _TL("Radius for local:"),
        _TL("Enter the local raidus value."),
        PARAMETER_TYPE_Int, 3, 0.0, true
    );
    
    Parameters.Add_Value(
        NULL, "CIV", _TL("Calculate intelligibility value:"),
        _TL("Option for the calculation of intelligibility value."),
        PARAMETER_TYPE_Bool, false
    );
    
    Parameters.Add_Value(
        NULL, "CRDL", _TL("Create network data:"),
        _TL("Option for the creation of network data."),
        PARAMETER_TYPE_Bool, false
    );
    
    Parameters.Add_Choice(
        NULL, "NW", _TL("Network data type:"),
        _TL("Choose a network data type for the output (UCINET dl file or PAJEK net file)."),
        CSG_String::Format(SG_T("%s|%s|"),
        _TL("UCINET's Data Language (dl) file"),
        _TL("PAJEK's Network (net) file")
        ), 0
    );
    
    Parameters.Add_Value(
        NULL, "CRGM", _TL("Create geodesic matrix:"),
        _TL("Option for the creation of geodesic matrix."),
        PARAMETER_TYPE_Bool, false
    );
    
    Parameters.Add_Value(
        NULL, "SFID", _TL("Saves file(s) in a different location:"),
        _TL("Option in order to save file(s) in a different location."),
        PARAMETER_TYPE_Bool, false
    );
Plugins and Scripts Developed to Conduct SSA in FOSS4GIS

Parameters.Add_Table_Field(
    pNode, "FIELD_1", _TL("Choose a field:"),
    _TL("Choose the field that will be used as the matrix headings in the output file."),
    false
);

Parameters.Add_Value(
    NULL, "UAF", _TL("Use a field:"),
    _TL("Option for the employment of a field as the matrix headings in the output file."),
    PARAMETER_TYPE_Bool, false
);

Actually the code given above is rather self-explaining without any further note. The name and the description of each variable parameter are set in the first line and a simple description of them is provided in the second line. After defining parameters window, the final step in the creation of the plugin is the production of the scripts mainly responsible for the construction of adjusted graph and calculation of standard space syntax measures. This is realized via On Execute() method. When the user defines the required information via GUI and presses the OK button in the parameters window, the method On Execute() is called (Olaya, 2004: 192). Nevertheless before writing the core codes, we should put the information entered by the user into some variables. Below is the part of the code produced for this purpose in SAGA SSA Plugin:

Script 19. Visual C++ code produced in order to define variables.

bool Cspace_syntax::On_Execute(void)
{
    bool bciv, bcrdl, bcrgm, bsfid, buaf;
    int pl, p2, s1, s2, sonuc, bonuc, row1, row2, k, l, sev, say, obj, liv, nNW;
    int cntr, iLineNo, iConnectivity, iTotalDepth, icntr, iMeanDepth, iGlobalInteg, iLocalDepth, iLocalInteg, iControl;
    int td, locd, deg3, ld3, cnt, Field_1;
    int* atr = NULL;
    int** sna = NULL;
    int** dst = NULL;
    double x1, y1, x2, y2, u1, v1, u2, v2, xi, yi, a1, a2, b1, b2, t1;
    double xmax, ymax, xmin, ymin, umax, vmax, umin, vmin;

    // Code goes here...
After defining variables, we can produce our scripts regarding SSA Plugin for SAGA. In this respect it should be emphasized that parallel to MapWindow there is a built-in function in SAGA GIS in order to check the intersection between the features in a vector layer. Nevertheless, the respective function could not be used in SSA Plugin for SAGA because of some problems experienced in using it (see http://sourceforge.net/mailarchive/message.php?msg_id=29097728). Because of this reason, intersection of features in SAGA is checked by employing the procedure described in Chapter 4 (Section 4.1.3. and Section 4.1.4.).

In addition to this, it is important to notice that if there is a multiline feature in a vector layer, SSA Plugin produced for SAGA only takes into account the first part of the respective feature without iterating over the other parts involved in a feature. This can easily be seen in Script 20 showing Visual C++ code produced for iteration of features in SAGA GIS in order to check intersection
between BBs. In the respective script, the coordinates ($e_1$ and $e_2$) of a multi-line feature is iterated over ($p_1$ and $p_2$ are counter variables) for only the first part of the feature (see $e_1 = pLine1->Get_Point(p1, 0)$ and $e_2 = pLine2->Get_Point(p2, 0)$).

Script 20. Visual C++ code produced for iteration of features in SAGA GIS in order to check intersection between bounding boxes.

```c++
obj = pL->Get_Count();
for(s1=0; s1<obj && Set_Progress(s1, obj); s1++)
{
    pLine1 = pL->Get_Shape(s1);
    row1 = pLine1->Get_Point_Count(0);
    e1 = pLine1->Get_Point(0, 0);
    xmax = e1.x;
    ymax = e1.y;
    xmin = e1.x;
    ymin = e1.y;
    for (p1=0; p1<(row1); p1++)
    {
        e1 = pLine1->Get_Point(p1, 0);
        x1 = e1.x;
        y1 = e1.y;
        if (x1 > xmax) xmax = x1;
        if (x1 < xmin) xmin = x1;
        if (y1 > ymax) ymax = y1;
        if (y1 < ymin) ymin = y1;
    }
}
for(s2=0; s2<obj; s2++)
{
    pLine2 = pL->Get_Shape(s2);
    row2 = pLine2->Get_Point_Count(0);
    e2 = pLine2->Get_Point(0, 0);
    umax = e2.x;
    vmax = e2.y;
    umin = e2.x;
    vmin = e2.y;
    for (p2=0; p2<row2; p2++)
    {
        e2 = pLine2->Get_Point(p2, 0);
        x2 = e2.x;
        y2 = e2.y;
        if (x2 > umax) umax = x2;
        if (x2 < umin) umin = x2;
        if (y2 > vmax) vmax = y2;
        if (y2 < vmin) vmin = y2;
    }
}
```
Operationalization of SSA in FOSS4GIS

```c
if (xmax < umin || ymin > vmax) bonuc = 0;
else if (xmin > umax || ymin > vmax) bonuc = 0;
else if (xmax < umin || ymax < vmin) bonuc = 0;
else if (xmin > umax || ymax < vmin) bonuc = 0;
else bonuc = 1;
```

In order to write the results of calculation to the attribute table, we should also define the new fields that will be added to the table. Although the respective part of SSA Plugin is not explained in a detailed context for other SSA Plugins, for SAGA the respective part of code is given as below;

Script 21. Visual C++ code produced to create new fields in the attribute table for space syntax parameters.

```c
// adding space syntax parameters as new fields to // the attribute table
pTable->Add_Field("LineNo", SG_DATATYPE_Int);
iLineNo = pTable->Get_Field_Count() - 1;
pTable->Add_Field("Connectivity", SG_DATATYPE_Int);
iConnectivity = pTable->Get_Field_Count() - 1;
pTable->Add_Field("TotalDepth", SG_DATATYPE_Int);
iTotalDepth = pTable->Get_Field_Count() - 1;
pTable->Add_Field("MeanDepth", SG_DATATYPE_Double);
iMeanDepth = pTable->Get_Field_Count() - 1;
pTable->Add_Field("GlobalInteg", SG_DATATYPE_Double);
iGlobalInteg = pTable->Get_Field_Count() - 1;
pTable->Add_Field("LocalDepth", SG_DATATYPE_Int);
iLocalDepth = pTable->Get_Field_Count() - 1;
pTable->Add_Field("LocalInteg", SG_DATATYPE_Double);
iLocalInteg = pTable->Get_Field_Count() - 1;
pTable->Add_Field("Control", SG_DATATYPE_Double);
iControl = pTable->Get_Field_Count() - 1;
```

5.9. Creating Scripts for R Project

Actually R is not a GIS software program. R is basically a free software environment for statistical computing and graphics. Nevertheless, we can read GIS files into R and perform calculations by using the scripting tools available in the respective environment. R scripts can be created by using simple text editors or “R editor” (Figure 40) that can be activated in R
via “File → New script”. Existing R scripts can also be opened via “File → Open script…”.

There is a rich collection of packages that can be considered as modules or plugins responsible for the calculation specific parameters or performing specific tasks in R. For example, there is already a package (*sna* package written by Butts (2010)) in R for the calculation of geodesics. Thus, once we read a vector layer into R and construct adjusted graph for the spatial configuration covered by the respective layer, we can easily calculate geodesics by employing the existing package available for this purpose in R.

![Figure 40. R editor to run and produce R scripts.](image)

Even there is another package (*rgeos*) in R for spatial geometry predicates and performing topology operations. The respective package, *rgeos* (Bivand and Rundel, 2012), implements functionality for the manipulation and querying of spatial geometries using GEOS library. Nevertheless, the script produced for SSA in R was created before the release of rgeos and it constructs adjusted graph for a given spatial configuration by checking the intersection between the features involved in a layer through BBs and iterating over the segments of polyline features rather than employing the built-in function already available in rgeos. Overall, SSA Script produced of R makes use of the following libraries (packages) in R:

- `library(shapefiles)`
- `library(tcltk)`
- `library(sna)`
- `library(maptools)`
shapefiles and maptools are required in order to read and process “.shp” files, tcltk is required in order to build a simple GUI and sna is used in order to calculate geodesics.

After reading a “.shp” file in R as shown in Figure 40, the script given in Figure 41 can be used in order to iterative over the features involved in the respective file and to check the intersection between BBs that are also retrieved by using the built-in method available for this purpose in R.

Figure 41. R script produced for iteration of features in a vector layer in order to check the intersection between bounding boxes.
6. How to Install and Use SSA Plugin, and How to Load and Process Network Data in SNA Software and Import Results Back to FOSS4GIS

A simple installation and user’s guide is prepared after finishing the plugins in order to share them with the community. This chapter is an expanded and revised version of the respective guide. Accordingly, in addition to the installation and use of SSA Plugin, users are also provided with the basic knowledge about how to open and process network files in several SNA software programs after they create the respective files by using SSA Plugin. Although user of the plugin is assumed to know basic GIS knowledge, the process through which tables (“.csv” or “.dbf”) created by using SNA software programs in order to calculate additional network parameters can be joined to the attribute table is also described in this chapter. SSA Plugins have been particularly designed for FOSS4GIS and released under the GNU GPL v2 license (see Appendix B) as it is also stated in the explanations given at beginning of the codes and the scripts for this purpose.

6.1. Installation of SSA Plugin

The procedure to install SSA plugin in FOSS4GIS is quite simple. Once we copy the files or folder listed in Table 7 and Table 8 to the directory where the plugins are located, we can directly begin to use SSA plugin without any difficulty. Nevertheless the user will suddenly notice that in some FOSS4GIS, SSA plugin could not be run or activated without loading or selecting a vector layer covering information about a spatial configuration for which an adjusted graph will be produced and subsequently space syntax parameters will be calculated.

After copying the files or folder to the location provided in Table 7 and Table 8, we should (re-)start FOSS4GIS we use. For OpenJUMP Jython SSA Plugin, existing startup.py and
mystartup.py are modified to work with the plugin. Thus, if we are familiar with these files, instead of copying them directly to the plugin folder we can modify the respective files by analyzing the files attached to the plugin. Further, if the location of startup.py and mystartup.py is different from the one given above, we should also modify the content of startup.py in order to properly run the plugin. Accordingly, we are required to modify “execfile("C:\Program Files\OpenJUMP\lib\ext\jython\mystartup.py")” part of startup.py file to define the location of both SSA Plugin and mystartup.py files.

In a similar fashion in Thuban, a thubanstart.py file is created in order to load SSA Plugin when the program starts. If we already have a thubanstart.py file, we should actually modify it by placing “import spacesyntax.py” inside it instead of copying thubanstart.py associated with the plugin to Thuban directory.

SSA plugin can be run in different FOSS4GIS via different menu options (see Table 9 for the menu system and toolbar through which we can activate and run SSA Plugin in different FOSS4GIS). In some FOSS4GIS (gvSIG, Quantum GIS, MapWindow GIS and OpenJUMP Jython) the plugin can also be run via a toolbar. The tool image for the plugin is as following:

SSA tool icon:  

In gvSIG there are two options for the plugin: (1) using Jython-Java GUI or (2) using xml GUI (see Figure 22 on page 64 for the images of GUIs). It is up to the prospective user which one to choose. The only difference is between GUIs. In a different fashion, there are three options to install SSA plugin in OpenJUMP: (1) BeanShell script version, (2) Jython plugin version and (3) Java (jar) version. Although BeanShell version is very slow compared with Jython and Java version, we may prefer to use the respective version in order to see and test how the scripts work. Jython version is again slow compared with Java version but it is easier and faster to produce scripts and plugins in Jython compared with Java for which we need a development platform and compiler such as Eclipse.
Table 7. SSA Plugin files or folder to copy and the location of plugin directory for various FOSS4GIS (Windows).

<table>
<thead>
<tr>
<th>FOSS4GIS</th>
<th>Files or folder to copy</th>
<th>Where to paste the files or folder</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenJUMP – BeanShell</td>
<td>Space Syntax Analysis.bsh</td>
<td>C:\Program Files\OpenJUMP\lib\ext\BeanTools</td>
</tr>
<tr>
<td>OpenJUMP – Jython</td>
<td>startup.py, mystartup.py,</td>
<td>C:\Program Files\OpenJUMP\lib\ext\jython</td>
</tr>
<tr>
<td></td>
<td>SpaceSyntax.py</td>
<td></td>
</tr>
<tr>
<td>OpenJUMP – Java</td>
<td>spacesyntax.jar</td>
<td>C:\Program Files\OpenJUMP\lib\ext</td>
</tr>
<tr>
<td>gvSIG – Jython GUI</td>
<td>SSA folder</td>
<td>C:\Program Files\gvSIG_1.9\bin\gvSIG\extensiones</td>
</tr>
<tr>
<td>gvSIG – xml GUI</td>
<td>SpaceSyntax folder</td>
<td>C:\Program Files\gvSIG_1.9\bin\gvSIG\extensiones</td>
</tr>
<tr>
<td>Thuban – Python</td>
<td>thubanstart.py, spacesyntax.py</td>
<td>C:\Program Files\Thuban</td>
</tr>
<tr>
<td>OpenEV – Python</td>
<td>spacesyntax.py</td>
<td>C:\Program Files\FWTools2.4.7\tools</td>
</tr>
<tr>
<td>Quantum GIS – Python</td>
<td>spacesyntax folder</td>
<td>C:\Program Files\Quantum GIS Wroclaw\apps\qgis\python\plugins</td>
</tr>
<tr>
<td>MapWindow GIS – VB.Net</td>
<td>Space Syntax folder</td>
<td>C:\Program Files\MapWindow\Plugins</td>
</tr>
<tr>
<td>SAGA GIS – C++</td>
<td>space_syntax.dll</td>
<td>C:\Program Files\SAGA-GIS\modules</td>
</tr>
<tr>
<td>OrbisGIS – BeanShell</td>
<td>Space Syntax Analysis.bsh</td>
<td>anywhere you like (you need to load the script)</td>
</tr>
<tr>
<td>R Project</td>
<td>Space Syntax Analysis.r</td>
<td>anywhere you like (you need to load the script)</td>
</tr>
</tbody>
</table>
Table 8. SSA Plugin files or folder to copy and the location of plugin directory for various FOSS4GIS (Ubuntu).

<table>
<thead>
<tr>
<th>FOSS4GIS</th>
<th>Files or folder to copy</th>
<th>Where to paste the files or folder</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenJUMP – BeanShell</td>
<td>Space Syntax Analysis.bsh</td>
<td>/usr/share/openjump/ext /BeanTools</td>
</tr>
<tr>
<td>OpenJUMP – Jython</td>
<td>startup.py, mystartup.py, SpaceSyntax.py</td>
<td>/usr/share/openjump/ext /jython</td>
</tr>
<tr>
<td>OpenJUMP – Java</td>
<td>spacesyntax.jar</td>
<td>/usr/share/openjump/ext</td>
</tr>
<tr>
<td>gvSIG – Jython GUI</td>
<td>SSA folder</td>
<td>../gvSIG/extensiones</td>
</tr>
<tr>
<td>gvSIG – xml GUI</td>
<td>SpaceSyntax folder</td>
<td>../gvSIG/extensiones</td>
</tr>
<tr>
<td>Thuban – Python</td>
<td>thubanstart.py, spacesyntax.py</td>
<td>../thuban</td>
</tr>
<tr>
<td>OpenEV – Python</td>
<td>spacesyntax.py</td>
<td>../tools</td>
</tr>
<tr>
<td>Quantum GIS – Python</td>
<td>spacesyntax folder</td>
<td>/usr/share/qgis/python/plugins/</td>
</tr>
<tr>
<td>OrbisGIS – BeanShell</td>
<td>Space Syntax Analysis.bsh</td>
<td>anywhere you like (you need to load the script)</td>
</tr>
<tr>
<td>R Project</td>
<td>Space Syntax Analysis.r</td>
<td>anywhere you like (you need to load the script)</td>
</tr>
</tbody>
</table>
Table 9. Menu items and toolbar through which SSA Plugin can be activated and run in various FOSS4GIS.

<table>
<thead>
<tr>
<th>FOSS4GIS</th>
<th>Menu path to run SSA plugin</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenJUMP – BeanShell</td>
<td>Customize → BeanTools → Space Syntax Analysis</td>
</tr>
<tr>
<td>OpenJUMP – Jython</td>
<td>Customize → Python Console and Tools... (activate Jython menu-tool) Tools → Space Syntax Analysis</td>
</tr>
<tr>
<td>OpenJUMP – Java</td>
<td>Graph Theoretic → Space Syntax Analysis</td>
</tr>
<tr>
<td>gvSIG</td>
<td>File → Scripting → Space Syntax Analysis</td>
</tr>
<tr>
<td>Thuban</td>
<td>Space Syntax Analysis → Space Syntax Analysis</td>
</tr>
<tr>
<td>OpenEV</td>
<td>Tools → Space Syntax Analysis</td>
</tr>
<tr>
<td>Quantum GIS</td>
<td>Plugins → Space Syntax Analysis → Space Syntax Analysis</td>
</tr>
<tr>
<td>MapWindow GIS</td>
<td>by clicking SSA Plugin icon or text on “SpaceSyntax” toolbar</td>
</tr>
<tr>
<td>SAGA GIS</td>
<td>Modules → Shapes → Space Syntax Analysis (or) Module Libraries → Space Syntax Analysis Module → Space Syntax Analysis</td>
</tr>
<tr>
<td>OrbisGIS</td>
<td>View → Beanshell Console (activate BeanShell Console) bsh script can be loaded and run via BeanShell Console</td>
</tr>
<tr>
<td>R Project</td>
<td>File → Open Script… (you can load and run R script)</td>
</tr>
</tbody>
</table>
Users may also experience some problems in installing and running the plugin for SAGA GIS. SSA Plugin works fine on the platform where I compile “.dll” file for SAGA, but I noticed that it may not work in other computers lacking the development environment. Lastly for OrbisGIS and R Statistical Package user are only provided with SSA scripts that should be manually loaded to the software each time they employ it.

Before running SSA Plugin, in some FOSS4GIS we are required to activate or load it. In this respect both in Quantum GIS and MapWindow GIS, we should first activate the plugin via the proper menu. In QGIS we can enable SSA Plugin via “Plugins → Manage Plugins…”. Once we active the plugin it can be run via both Menu bar and Plugin toolbar. In MapWindow GIS, we can activate SSA Plugin icon via directly selecting “Plug-ins → Space Syntax Analysis” or “Plug-ins → Edit Plug-ins” where we can get some extra information about the plugin (such as author and version number). In some FOSS4GIS, plugin can be run after activating another tool. For example in OpenJUMP in order to run Jython SSA plugin we are first required to activate Jython menu and tool bar, and then we can select SSA via either “Tools - Space Syntax Analysis” or SSA text icon placed on the Jython toolbar.

### 6.2. Use of SSA Plugin

Once we select the menu item or click the toolbar icon provided for SSA Plugin, a dialog box appears in the window as a Graphical User Interface (GUI) in order to make it possible for us to activate the options in relation to the outcomes of space syntax analysis in addition to the calculation of basic space syntax parameters. If no option is selected via GUI (Figure 42), plugin assigns a sequential number to each feature (currently lines or regions) and calculates the following parameters for a given spatial configuration (see Chapter 3 for the elaboration of the respective parameters, and Chapter 4 and Chapter 5 for the conceptualization and operationalization of them as a SSA plugin in FOSS4GIS):
Plugins and Scripts Developed to Conduct SSA in FOSS4GIS

Parameters calculated by SSA Plugin:

For each feature in a vector layer;

- Connectivity,
- Total Depth,
- Mean Depth,
- Global Integration,
- Local Depth,
- Local Integration,
- Control Value.

For the whole spatial configuration;

- Intelligibility value

Figure 42. Parameters calculated by SSA Plugin and its GUI.

If user checks the option “Calculate intelligibility value”, in addition to the above parameters plugin also calculates the Pearson Correlation (i.e. intelligibility value) between connectivity and global integration. For those who are willing to calculate graph theoretic parameters other than the ones available in a standard SSA software including the ones offered in this plugin, extra option is available to transfer the adjusted graph created by the plugin to an external file (Ucinet’s “.dl” or Pajek’s “.net” format) that can be further processed in the software programs specifically designed for social network analysis (SNA) (such as Cytoscape, Gephi, Tulip, Network Workbench (NWB), Social Network Visualiser (SocNetV), visual social networks (visone), Pajek, Organizational Risk Analyzer (ORA) and UCINET). For this purpose, we should mark “Create network data:” option that makes it possible for us to choose a file format to save the resulting adjusted graph.
How to Install and Use SSA Plugin

Once we save the adjusted graph in a SNA file format, we can easily open the respective network data, for example, in Cytoscape and conduct further analysis (such as calculation of other measures of centrality including closeness, betweenness, radiality, stress and eccentricity). The new parameters calculated in Cytoscape can later be transferred to FOSS4GIS by saving the results to a “.csv” or “.txt” file that can easily be linked to the original spatial data in GIS via the proper procedure available to connect an external tabular data to the existing attribute table.

We can also write the geodesic matrix to a text (“.txt”) file if we mark the option for “Create geodesic matrix.” Once the user chooses any of the options available for the creation of an external file that can be used in other software programs particularly developed for social network analysis, plugin also asks for the matrix headings that will appear in the output files. In this respect, two options are available for the matrix headings:

(1) employment of the internal index that range between 1 and the number of features involved in the network,

(2) employment of a field in the attribute table as the matrix headings.

What is particularly important for the second option is the fact that the field chosen for the matrix heading should not include duplicate IDs (identity numbers) for (line or polygon) features. If we choose to use a field from the attribute table, we are asked to provide the respective field. Irrespective of the users’ choice, the internal IDs of the features (Lineno) are also added to the new attribute table in addition to the space syntax parameters in order to make it possible for the users to re-integrate the parameters calculated outside FOSS4GIS to the attribute table.

If the option available to “Save the file(s) in a different location” is checked, we are asked to provide location and name of the file that will store the adjusted graph or geodesic distance. Otherwise, the files created for adjusted graph and geodesic distance are saved in the directory where the source layer is
Plugins and Scripts Developed to Conduct SSA in FOSS4GIS

located by using the name of the source layer if the respective options are marked to produce output files.

SSA Plugin GUI given in Figure 42 is almost standard for all FOSS4GIS for which a SSA plugin or script is developed by the author except for Thuban and R Project where user is additionally asked for to provide a file name to which the output GIS file (“.shp”) will be written (see Appendix A for the images of GUIs of SSA plugin and scripts in various FOSS4GIS). Except for Thuban and R Project it was possible for the author to write the results of the analysis to either the attribute table of the layer selected to be analyzed or to a new layer (gvSIG only). However, in other FOSS4GIS users should also save the results of the analysis before they close FOSS4GIS they prefer to use. Otherwise, all the new information can be lost. In R, when the script is run, it first asks for the location of “.shp” file to be processed. After selecting “.shp” file, script constructs the adjusted graph and calculates geodesics by using sna package written by Butts (2010). After these calculations an option is also offered to the user via SSA GUI to calculate intelligibility value and to enter the radius for local value. And when we press OK, we are prompted to provide the location and name of the file to which the results of the analysis (parameters) will be written. Considering the existence of a social network analysis library (sna) in R Project, no option is created to export the adjusted graph to an external file in the respective software program. After running the script, user will already have the adjusted graph stored in a two dimensional array named as ‘sna’. By using sna library we can easily calculate, for example, betweenness and closeness centralities (just type betweenness(sna) or closeness(sna) in the console and press enter). We can also calculate other graph theoretic parameters available in sna library for the spatial configuration concerned.

We can import any spatial data produced by employing computer assisted drawing (CAD) software programs to FOSS4GIS by converting the respective data first into a “.dxf” (drawing exchange format) file and then importing the resulting file to FOSS4GIS that we prefer to use via the module or option available for the respective task. SSA plugin for FOSS4GIS are particularly designed to be operational for the exchange of data
between GIS and SNA software programs in which a wide range of network parameters can easily be calculated thanks to their specific focus on graph theoretical analysis.

Although users of the plugin are assumed to know the basic knowledge about GIS, it would be helpful to describe how they can produce thematic maps (see Figure 43 – 51 for some examples from Siteler, an industrial cluster in Ankara) in GIS by using the parameters calculated by SSA Plugin in each FOSS4GIS. In this respect, Table 11 shows the menu items or mouse button through which we can produce thematic maps for different fields (parameters) available in an attribute table.

Lastly, as this chapter is not intended to provide users with some knowledge about how to use a GIS software program, if users are unfamiliar with GIS programs, they will need to search for a simple user’s guide for FOSS4GIS they prefer to use in order to, at least, to open or load a vector layer into FOSS4GIS. Nevertheless, most of the time the respective procedure is very simple for the majority of FOSS4GIS and we can easily explore how to add a vector layer to the program. We may obtain information about a general user guide for FOSS4GIS we prefer to use via the following links (Table 10);

Table 10. Links for FOSS4GIS on which SSA Plugin is operational.

<table>
<thead>
<tr>
<th>FOSS4GIS</th>
<th>link</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenJUMP</td>
<td><a href="http://www.openjump.org/">http://www.openjump.org/</a></td>
</tr>
<tr>
<td>gvSIG</td>
<td><a href="http://www.gvsig.org/web/">http://www.gvsig.org/web/</a></td>
</tr>
<tr>
<td>Thuban</td>
<td><a href="http://thuban.intevation.org/">http://thuban.intevation.org/</a></td>
</tr>
<tr>
<td>OpenEV</td>
<td><a href="http://openev.sourceforge.net/">http://openev.sourceforge.net/</a></td>
</tr>
<tr>
<td>Quantum GIS</td>
<td><a href="http://www.qgis.org/">http://www.qgis.org/</a></td>
</tr>
<tr>
<td>MapWindow</td>
<td><a href="http://www.mapwindow.org/">http://www.mapwindow.org/</a></td>
</tr>
<tr>
<td>OrbisGIS</td>
<td><a href="http://www.orbismgis.org/">http://www.orbismgis.org/</a></td>
</tr>
<tr>
<td>R Project</td>
<td><a href="http://www.r-project.org/">http://www.r-project.org/</a></td>
</tr>
</tbody>
</table>
Table 11. Menu and mouse button path to produce thematic maps in various FOSS4GIS.

<table>
<thead>
<tr>
<th>FOSS4GIS</th>
<th>Menu and mouse button path to produce thematic maps</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenJUMP</td>
<td>Press the right button of the mouse while the cursor is on the name of the layer, and then select “Change Styles → Colour Theming”. Check “Enable colour theming” and choose among the options available for both “Classification Method” and “Attribute” (i.e. parameters).</td>
</tr>
<tr>
<td>gvSIG</td>
<td>Press the right button of the mouse while the cursor is on the name of the layer, and then select “Properties… → Symbology → Quantities → Intervals”. Choose among the options available for both “Interval type” and “Classification field” (i.e. parameters).</td>
</tr>
<tr>
<td>Thuban</td>
<td>“Layer → Properties”, and then select a “Field” (parameter) and press “Generate Class”</td>
</tr>
<tr>
<td>OpenEV</td>
<td>On the toolbar click “Classify Layer” tool, and then you can select the field (parameter) for thematic map. Press “reclassify…” in order to choose among the options available for classification “Type”.</td>
</tr>
<tr>
<td>Quantum GIS</td>
<td>“Layer → Properties…”, and then select “Style” tab and “Graduated” option for the classification. Select the parameters from the available “Column”s.</td>
</tr>
<tr>
<td>MapWindow GIS</td>
<td>“Layer → Properties”, and then select “Categories” tab and uncheck “Unique values”. Select the parameters from the available “Fields” and press “Generate” in order to classify the values.</td>
</tr>
<tr>
<td>SAGA GIS</td>
<td>Activate “Show Object Properties” via “Window” menu. And then select “Colors → Type → Graduated Color” and define the parameter via “Colors → Type → Graduated Color → Attribute”.</td>
</tr>
<tr>
<td>OrbisGIS</td>
<td>Press the right button of the mouse while the cursor is on the name of the layer, and then select “Edit Legend”. And then press “Add” tool and select “Interval classification”. Define “Classification field;” (parameter) and “Type of interval;”.</td>
</tr>
<tr>
<td>R Project</td>
<td>Script automatically produces a thematic for the global integration values. Please refer to R manuals and also analyze the script to see how to produce alternative thematic maps.</td>
</tr>
</tbody>
</table>
How to Install and Use SSA Plugin

Figure 43. A thematic map produced in OpenJUMP for Global Integration values.

Figure 44. A thematic map produced in Quantum GIS for Global Integration values.
Plugins and Scripts Developed to Conduct SSA in FOSS4GIS

Figure 45. A thematic map produced in Thuban for Global Integration values.

Figure 46. A thematic map produced in OpenEV for Global Integration values.
How to Install and Use SSA Plugin

Figure 47. A thematic map produced in gvSIG for Global Integration values.

Figure 48. A thematic map produced in SAGA GIS for Global Integration values.
Plugins and Scripts Developed to Conduct SSA in FOSS4GIS

Figure 49. A thematic map produced in MapWindow GIS for Global Integration values.

Figure 50. A thematic map produced in OrbisGIS for Global Integration values.
6.3. Loading Adjusted Graph into Various SNA Software Programs and Calculation of New Parameters

Once a network file is created for the adjusted graph by selecting the available options via GUI of SSA Plugin, the respective file can be opened in a wide range of open source or free software programs particularly created for SNA. A list of network analysis software programs for which a description is provided about how to open “.dl” or “.net” files can be seen in Table 12. Menu paths for opening or importing “.dl” or “.net” files into the respective programs are also given in Table 13 (for “.net” files) and Table 14 (for “.dl” files).

After opening or importing the adjusted graph into the software programs listed in Table 12 by following menu paths given in Table 13 or Table 14, a series of graph theoretic parameters can be calculated by following the procedures described below for each program;

Figure 51. A thematic map produced in R Project for Global Integration values.
Plugins and Scripts Developed to Conduct SSA in FOSS4GIS

Table 12. Types of software licenses for various network analysis software programs.

<table>
<thead>
<tr>
<th>SNA Software</th>
<th>Licenses</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cytoscape</td>
<td>GNU LGPL</td>
<td>Version 2.8.3</td>
</tr>
<tr>
<td>Gephi</td>
<td>dual licensed under CDDL and GNU GPL</td>
<td>Version 0.8.1 beta</td>
</tr>
<tr>
<td>Tulip</td>
<td>GNU LGPL</td>
<td>Version 3.8.0</td>
</tr>
<tr>
<td>Network Workbench (NWB)</td>
<td>Apache License</td>
<td>Version 1.0.0 official</td>
</tr>
<tr>
<td>Social Network Visualiser (SocNetV)</td>
<td>GNU GPL</td>
<td>Version 0.81</td>
</tr>
<tr>
<td>visual social networks (visone)</td>
<td>Free for academic and research purposes</td>
<td>Version 2.6.5</td>
</tr>
<tr>
<td>Pajek</td>
<td>Free for non-commercial use</td>
<td>Version 3.03</td>
</tr>
<tr>
<td>Organizational Risk Analyzer (ORA)</td>
<td>Free for academic and research purposes</td>
<td>Version 2.2.8.b</td>
</tr>
<tr>
<td>UCINET</td>
<td>Shareware</td>
<td>Version 6.403</td>
</tr>
</tbody>
</table>

Note: General Public License (GPL), Lesser General Public License (LGPL), Common Development and Distribution License (CDDL)
Table 13. Loading adjusted graph saved in “.net” file format into various network analysis software programs.

<table>
<thead>
<tr>
<th>SNA Software:</th>
<th>How to open network file: Menu path</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pajek</td>
<td>“File → Network → Read → Pajek networks (*.net)”</td>
</tr>
<tr>
<td>UCINET</td>
<td>“Data → Pajek”, and then define “Input Pajek file” by browsing via “…”</td>
</tr>
<tr>
<td>ORA</td>
<td>“File → Data Import Wizard... → Import existing data? → Data from another network analysis tool → Pajek → Next → Browse”</td>
</tr>
<tr>
<td>Cytoscape</td>
<td>You should repeat the same procedure defined for opening “.dl” file by only entering total number of features plus 1 instead of 4 in “Attribute Names” section for “Transfer first line as attribute names Start Import Row:”. If you successfully complete the process to load the adjusted graph into the program, you will see an information window: “Loading Network and Edge Attributes”.</td>
</tr>
<tr>
<td>Gephi</td>
<td>“File → Open”, select “Net Files (Pajek) (*.net)” for “Files of type”, and then by accepting default option (“New graph”) press “OK”</td>
</tr>
<tr>
<td>Tulip</td>
<td>“File → Import → File → Pajek (.net)” (net option does not work properly)</td>
</tr>
<tr>
<td>NWB</td>
<td>“File → Load…”, and then choose a Pajek .net file</td>
</tr>
<tr>
<td>Visone</td>
<td>“File → Open → Pajek graph files (.net)”</td>
</tr>
<tr>
<td>SocNetV</td>
<td>“Network → Import → Pajek”</td>
</tr>
</tbody>
</table>
Table 14. Loading adjusted graph saved in “.dl” file format into various network analysis software programs.

<table>
<thead>
<tr>
<th>SNA Software</th>
<th>How to open network file: Menu path</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pajek</td>
<td>“File → Network → Read → UCINET DL files (*.dat)” (gives “Error in vertex label!”)</td>
</tr>
<tr>
<td>UCINET</td>
<td>“Data → Import - DL…”, and then define “Input text file in DL format” by browsing via “…”</td>
</tr>
<tr>
<td>ORA</td>
<td>“File → Data Import Wizard… → Import existing data? → Data from another network analysis tool → UCINET (text or binary) → Next”; after next, define a name for the network via “Create a new meta-network with ID:” and then “Next → Browse” for “.dl” file. For “Source type:” and “Target type:” select “Agent”, and press “Finish”.</td>
</tr>
<tr>
<td>Cytoscape</td>
<td>“File → Import → Network from Table (Text/MS Excel)…”, and then for “Input File” click “Select File(s)” and choose ”.dl” file. After pressing “Open”, you can see the first 100 rows of the text file. In order to properly import the data you should mark “Show Text File Import Options” in “Advanced” section, enter 4 in “Attribute Names” section for “Transfer first line as attribute names Start Import Row:” and mark the respective option. When you press “Refresh Preview”, you will see that “Column1” and “Column2” are more properly listed. Lastly, in section “Interaction Definition” for “Source Interaction” choose “Column 1” and for “Target Interaction” choose “Column 2” without modifying the default option for “Interaction Type” (i.e. “Default Interaction”). When you finish all these steps, press “Import”.</td>
</tr>
<tr>
<td>Gephi</td>
<td>“File → Open”, select “DL Files (UCINET) (*.dl)” for “Files of type”, and then by accepting default option (“New graph”) press “OK”.</td>
</tr>
<tr>
<td>Tulip</td>
<td>“File → Import → File → UCINET dl”</td>
</tr>
<tr>
<td>NWB</td>
<td>There is no option to load “.dl” file, but you can load it by deleting the first 4 lines inside “.dl” file and saving it as a text file. Later, “File → Load…”, and then choose the respective text file. In “Load” window, select “Load as…” “Edge List”.</td>
</tr>
<tr>
<td>visone</td>
<td>“File → Open → UCINET .dl files (.dl)”</td>
</tr>
<tr>
<td>SocNetV</td>
<td>“Network → Import → DL…” (dl option does not work properly)”</td>
</tr>
</tbody>
</table>
How to Install and Use SSA Plugin

Analysis of Adjusted Graph in Cytoscape

After loading the adjusted graph into Cytoscape, we can analyze the network data via “Plugins → Network Analysis → Analyze Network” in order to calculate some parameters (see also Shannon et al. (2003) and Smoot et al. (2011) for the respective software). We can see the respective parameters by selecting “Filters” tab and then clicking “Select All”. There (“Data Panel” window) we can notice that MeanDepth is recalculated as “AverageShortest PathLength”. In “Data Panel” we can see the calculated parameters for each BSU. If we right-click the mouse on any cell, we can see that there is an option to “Export...” “Entire Table”. We can export the results to a “.txt” file by using this option. For further graph theoretic parameters we can navigate through “Plugins → Manage Plugins” and choose the plugin that can calculate the parameters for which we want to produce thematic map.

Analysis of Adjusted Graph in Gephi

After loading the adjusted graph into Gephi, if it is not activated, we should first activate statistics window via “Window → Statistics” and also table view via “Window → Data Table” in order to see the results of the analysis in the table (see Bastian et al. (2009) for the respective software). From “Statistics” tab, “Run” for example “Network Diameter” and choose “Undirected” and press “OK”. If we activate “Data Table”, we will see that new columns are added to the table (Eccentricity, Closeness Centrality and Betweenness Centrality). We can also calculate other parameters by simply “Run”ning other parameters. After finishing the calculation of the parameters, we can export it to a text file via “Export table” menu available at the top of “Data Table”. Once we press “Export table”, we will be prompted to define “Separator”, “Charset” and the “Columns” that will be included in the output file. Then press “OK” to create output text file.
Analysis of Adjusted Graph in Tulip

After opening adjusted graph in Tulip, we can calculate some parameters (such as “Betweenness Centrality”, “Eccentricity” and “Strength”) via “Algorithm → Measure → Graph” (see Auber (2003) and Auber et al. (2012) for the respective software). In “Tulip Parameter Editor”, select “viewMetric” (default option) for “result”. After calculating any parameter, we can see the table storing the values of the chosen parameter via “View → Table view”. In the respective table, “viewMetric (Metric)” field shows the values of the calculated parameter for each BSU. It seems that there is no option to properly export the results of analysis to a “.csv” file on Tulip. Nevertheless, the results of the analysis can be written to a “.tlp” file via “File → Export → tlp”. After exporting the results to “.tlp” file, it can be opened and edited in order to derive the parameters calculated. The section given below in tlp file includes the parameters calculated for each BSU. The respective section can be edited in order to create a “.csv” file that can easily be transferred to a GIS file.

```
(property 0 graph "viewMetaGraph"
(default "" "()"
)

(property 0 double "viewMetric"
(default "0" "0"
)(node 0 "425.561")
```

Analysis of Adjusted Graph in Network Workbench Tool (NWB)

After loading the network file into NWB (NWB Team, 2006), we can see the adjusted graph via “Visualization → GUESS” or “Spring(JUNG)”. We can also calculate some parameters via “Analysis” menu (such as “Unweighted and Undirected → Node Degree”). Once we choose a parameter to calculate, a new item is created in “Data Manager”. When we press the right mouse button while the cursor is over the new item, we can choose to view the results in a text editor in different formats (such as Pajek “.mat” and “.net” formats). After editing the text (converting into “.csv” format or saving it in “.dbf”), it can be transferred to GIS.
Analysis of Adjusted Graph in Social Network Visualiser (SocNetV)

After importing adjusted graph into SocNetV, via “Statistics → Centralities” we can, for example, calculate some centrality measures (see Kalamaras (2010) for the respective software). After calculating any parameter, we can edit the result in order to save it in a “.txt” file format that can be imported into GIS. For this purpose, it is necessary to delete the information given in “.dat” file both at the beginning and end of the table. Only table with node ids and parameters together with column headings should be kept to be processed in Open Office to produce a “.csv” file or save in “.dbf” format.

Analysis of Adjusted Graph in visone

After opening the network data in visone, select “Analysis” tab and then select “indexing” for “task:”, select “node centrality” for “class:”, select “betweenness” for “index:” and then press “analyze!” button at the bottom (see Brandes and Wagner (2004), and Baur (2008) for the respective software). After calculating the parameter, in “selection” tab press “attribute manager” that activates a window where we can see the value of parameter for each BSU in the form of table. In the respective window, we should first set that we are dealing with “nodes” at the top of the window and later define “values” to export (choose first “id” and then parameters to be included in the text file such as “betweenness”). And then select “import & export”. In the new window choose “export as table” or only “export”; and then define location and name of our text file for the output by setting “sort by” (default) to “id”. In “save options”, select “user defined” for “preset”, select “TAB” for “cell delimiter” and select “NONE” for “textframe”. The resulting tab separated text file can easily be opened both in GIS and Open Office to save in other formats such as “.dbf”.

117
Analysis of Adjusted Graph in Pajek

After reading the network data, we can, for example, calculate some centrality measures such as betweenness and closeness via “Net → Vector → Centrality → Betweenness” or “Closeness → All” (or “Input” or “Output”) (see Batagelj and Mrvar (2011) for the respective software). After the calculation, we can see that the result of the analysis is added to “Vectors”. If we press “Edit Vector” tool button, we can see the table storing results. We can save the results in Pajek “.vec” format. If we try to calculate degree centrality measure (via “Net → Partitions → Degree → All” or “Input” or “Output”), we will notice that the result is added as a partition to “Partitions”. These are program specific characteristics of Pajek. For “.vec” file we should add a sequential number to each row in order to associate the results with GIS data. Thus, it is a little bit tricky to import the parameters calculated in Pajek back to GIS.

Analysis of Adjusted Graph in ORA (Organizational Risk Analyzer)

After loading the network data into ORA, we can select among the parameters that will be calculated for the nodes via “Analysis → Measures Manager...” (see Carley et al. (2011) for the respective software). It would be better first to activate “Un-Select Visible” and then to select the parameters from the list provided (by grouping them via “Last Name” for example “Centrality”). After defining parameters, we can calculate parameters via “Analysis → Generate Reports → Show me everything (All Measures)”. In the coming window, we should select “Meta-Network” we have defined when we have imported the data to the program; and then press “Next”. After pressing “Next”, we will be asked for the options to calculate the parameters for the overall network or each node in the network. When we press “Next” after selecting among the options, we will be able to select among the formats for the output.

If we select csv (comma separated values) and define a proper directory and file name, results of the analysis will be saved in the file according the node labels we defined while we export
adjusted graph into network data format. It is important to notice that there are some extra lines at the end of the report file (".csv") (such as "GINI-COEFFICIENT" and "HERFINDAHL-INDEX"). It would be helpful to delete respective lines before importing the results of analysis back to GIS because they may create problems in some FOSS4GIS during importing process.

**Analysis of Adjusted Graph in Ucinet**

After creating ".##h" format Ucinet data (Borgatti, Everett and Freeman, 2002), we can easily conduct network analysis and calculate new parameters for the network data (see also Hanneman and Riddle (2005) for the respective software). For example, via “Network → Centrality”, we can choose among different centrality measures and calculate the respective measures. Values of calculated parameters for each BSU can be seen in the window “Output Log”. Although it is possible to save the output log by arranging it and saving it as a text file, since it is in fixed width format it can not easily be transferred to GIS. Thus, via “Data → Spreadsheets → Matrix” and “File → Open”, output ".##h" file can be opened in Ucinet; and then via “Edit – Select all” we can copy the results to a text editor in order to save it in tab separated format that can easily be read by many FOSS4GIS. Alternatively the resulting files can also be saved in ".dbf" format by using Open Office.

6.4. Importing Parameters Calculated in SNA Software Programs Back to FOSS4GIS

After calculating some graph theoretic parameters outside FOSS4GIS by using various SNA software programs listed in the previous section, we can import them back to GIS in order to create some thematic maps or conduct further spatial analysis based on the new parameters. In majority of FOSS4GIS there is an option to link external tables to the attribute table of the layer we are working on. Table 15 shows short menu path for opening external tables or linking them to an existing attribute table in various FOSS4GIS for which a SSA Plugin is created.
Table 15. Importing tabular data into FOSS4GIS.

<table>
<thead>
<tr>
<th>FOSS4GIS</th>
<th>Menu path to import tabular data:</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenJUMP</td>
<td>Tools → Edit Attributes → Join TXT table….</td>
</tr>
<tr>
<td>gvSIG</td>
<td>in “Project manager” for “Table” select “New”</td>
</tr>
<tr>
<td>Thuban</td>
<td>Table → Open</td>
</tr>
<tr>
<td>OpenEV</td>
<td>no option</td>
</tr>
<tr>
<td>Quantum GIS</td>
<td>Layer → Add vector layer… → File</td>
</tr>
<tr>
<td>MapWindow</td>
<td>Swift-D → Import External Data</td>
</tr>
<tr>
<td>OrbisGIS</td>
<td>right-click in the empty area of “Geocatalog” and then select “Add → File”</td>
</tr>
<tr>
<td>SAGA GIS</td>
<td>File → Table → Load</td>
</tr>
</tbody>
</table>

If we save the results of the analysis conducted in SNA software programs in “.csv” or “.txt” format, we are usually required to define a delimiter used in the table to separate fields from each other in order to import the respective table into GIS. This procedure is almost typical of all FOSS4GIS. Once we attempt to link an external table to the attribute table of a selected layer, we are required to define ID fields that will be used in order to associate the records available in the external table to the records available in the attribute table of the selected layer. In the remaining parts of this section, the respective procedure is defined in a detailed context for each FOSS4GIS over which SSA Plugin is operational.

For OpenEV and R Project no explanation is provided for importing tabular data, as there is no such kind of an option in OpenEV and in R Project users can easily calculate other graph theoretic parameters by using existing packages (such as sna) without exporting adjusted graph into SNA software programs.
Importing Tabular Data into OpenJUMP

In OpenJUMP, we can associate “.csv” files with the attribute table of a layer via “Tools → Edit Attributes → Join TXT table...”. When we select the respective menu item, we will be required to “Choose file data to join”. After choosing a file, we will be required to define “Layer” to which the chosen text file will be joined. In the coming window named as “Matching Fields”, we should define “Layer Field” (field in the selected layer) and “Table Field” (corresponding field in the table covering parameters) in order to join the table with the attribute table of the selected layer.

After joining text file with the attribute table, we can see the new parameters in the attribute table by right clicking the name of the selected layer and choosing “View / Edit Attributes”.

Importing Tabular Data into OrbisGIS

In OrbisGIS, we should first activate “Geocatalog” via “View → Geocatalog”. After activating “Geocatalog”, right-click in the empty area of “Geocatalog” and then select “Add → File”, for “Files of type:” choose either “.csv” or “.dbf”. After opening the external table, we can see it in “Geocatalog”. Lastly, we should conduct the following SQL query in order to create a new layer with the attributes of both existing attribute table and external table:

```sql
create table fer as select * from gisfile a, snafile b
WHERE a.FID = b.KID;
```

where `gisfile` is the name of layer covering spatial configuration and `snafile` is the name of the file including parameters calculated in a SNA software program. `FID` and `KID` stand for, respectively, identity fields in `gisfile` and `snafile`. After executing SQL query, we will have a new layer with name `fer` for which we can create new thematic maps using the parameters involved in `snafile`. 
Importing Tabular Data into gvSIG

In gvSIG, in “Project manager” we should first choose to create “New” table. And then using “File” tab, “Add” any “.dbf” or “.csv”. It is important to note that for “.csv” files, separator should be compatible with the one used in the system. After adding a vector layer to “View”, selected layer’s attribute table can be joined to the external table as following:

We should first “Show attribute table of active layer” (for this purpose click the respective tool on the toolbar – tool tip can help you find the respective tool while you are slowly surfing over them). While the attribute table is selected, activate “Table → Join” and for “Name:” select the attribute table and for “Field to join on:” select the identity field that will be used in order to associate registers in the external table to the existing attribute table. Press “Next” and in the coming window regarding “Import fields from table:”, for “Name:” select the newly added “.csv”, “.txt” or “.dbf” table, for “Field to join on:” select the identity field corresponding to the identity field selected for the attribute table and press “Finish”. Now we have other parameters added to the attribute table and we can produce thematic maps for them.

Importing Tabular Data into Thuban

In Thuban, we should save the table produced for the new parameter table in “.dbf” format. And then it is possible to open the respective table via “Table → Open”. We can join this external table with the attribute table of a selected layer via “Layer → Join Table…”. In the coming window (“Join Layer with Table”), we should first select “Field:” for the attribute table, and then select newly created “Table:” to join and “Field:” of association in the respective table. After finishing this procedure press “Join” to join the external table with the attribute table of the selected layer. If we right-click the name of the layer and select “Show Table”, we can see that new parameters are added to the attribute table.
How to Install and Use SSA Plugin

Importing Tabular Data into Quantum GIS

In QGIS, we should first open table by simple choosing “Layer → Add vector layer... → File” for “Encoding” select the most appropriate one. Then, “Browse” for the table (“.csv”, “.txt” or “.dbf”). After opening table we can right-click the name of the table in “Layers” and “Open attribute table”.

The last step is to join this newly opened table to the attribute table of the existing vector layer. For this purpose, we should right-click the selected vector layer and select “Properties” from coming menu. And then we should activate “Joins” tab there. If we press “+”, we will be prompted to define “Join layer” for which we should select the table. “Join field” is the field that will be used in order join the external table and “Target field” is the corresponding field in the attribute table. Once we press “OK”, we will see that the table is added to the view (Join layer - Joint field - Target field). The last step is to press “Apply”. When we right-click the selected vector in order to “Open attribute table”, we can see that the fields in the external table are added to the attribute table as new fields.

Importing Tabular Data into SAGA

In SAGA GIS, we should first open the external table via “File → Table → Load”. After loading it, we can see the content of the table via “Data → Tables”. In order to join the respective table with the attribute table of the selected layer, we should follow “Modules → Shapes → Tools → Join a Table” in menu bar or via Modules view we should select “Modules → Shapes-Tools → Join a Table”. In the coming window (“Join a Table”), we should first select “Shapes” (layer), “Identifier” (field of association in the attribute table), and define “Resulting Shapes” as “[create]”. And then we should select “Table” (external table), “Identifier” (field of association in the external table). Lastly, we should define options for “Attributes” as “append” and “Copy Rule” as “all shapes”, press “Okay”. Now we will see that a new shape file is created under “Data → Shapes → Line”. 

123
Importing Tabular Data into MapWindow GIS

In MapWindow GIS, we can join an external table to the attribute table of the selected layer by using Swift-D Plugin developed by Dinesh Grover (2012). After activating the respective plugin via “Plug-ins → Swift-D”, we should run the option of join via “Swift-D → Import External Data”. Then we should define the external table by selecting the type of the table (“.dbf”, “.xls” or “.mdb”). After selecting the table to process, we can see the fields available in it under “Imported Data”. In order to join the external table with the attribute table, we should define the fields that will be used to associate the data between two tables by selecting the respective fields under “Step 2: Select and Highlight Key Field(s)” for both “Shapefile” and “Imported Data”. In “Step 3: Select Column to be Imported”, unfortunately we are allowed to join-import only one column at a time. We can add new parameters to the attribute table by pressing “Step 4: Attach”. After adding the new parameters to the attribute table, we can produce thematic maps and conduct further analysis by using them.
7. General Evaluation and Conclusion

There is no doubt that FOSS movement has opened the door for GIS community to develop and share quality geo-informatics software programs as it was the case for the earlier paths along which the movement made it possible for the other communities to develop quality software programs in different fields. In terms of advanced GIS functions, properties and processes, FOSS4GIS are becoming more and more competitive compared with the closed source software programs. Today, there exists a rich collection of libraries for further development of FOSS4GIS. Moreover, they are becoming increasingly available to a larger group of users all over the world. FOSS4GIS saves us to reinvent the wheel and makes it possible to build upon the experience of the others.

Within this context, after unveiling the rational behind the preference for FOSS in the development of SSA Plugin, in the book the reader has been first introduced to the basic concepts and definition about SSA in connection with the need to exchange data between SSA and SNA software programs. Accordingly, the concept of ‘adjusted graph’ and standard SSA parameters have been elaborated in connection with SNA together with the options available for the construction of line networks and the necessity of interaction between SSA Plugin and SNA software programs. After elaborating the basic concepts and definitions in SSA, the conceptualization and structure of SSA scripts have been presented by drawing on the algorithms and sample scripts produced for certain parts of SSA.

It has been shown that in terms of the spatial databases that can be processed by a standard GIS software program, there are alternative approaches for the graph theoretical analysis of the spatial configurations by using SSA (such as employing vector layers as the base to represent any spatial configuration (for example streets, buildings or units of a building) or using image layers (particularly grid images) as the base for the representation of the spatial configurations). It has also been demonstrated that
the core of SSA script is actually composed of three basic sections; (1) the construction of adjusted graph showing the adjacency between BSUs, (2) the calculation of geodesics between BSUs, and (3) the calculation of the standard space syntax parameters. Nevertheless, it has been illustrated that there are also other functional parts of SSA Plugin that have been integrated into other parts.

Particularly, in the fifth chapter of the book, the establishment and employment of the scripting platforms required for the creation of SSA Plugins in various FOSS4GIS have been illustrated by drawing on the particular case of SSA Plugins. As SSA Plugins are composed of certain standard sections elaborated in Chapter 4, in Chapter 5 mainly the procedure to iterate over the features involved in a vector layer has been presented together with the establishment of minimum software environment required to create a script or plugin that can be used for analyzing vector layers. In the sixth chapter an extended version of the installation and user’s guide for SSA Plugin has been presented by particularly drawing on the exchange of data between GIS and SNA software programs via SSA plugins and the tools available in FOSS4GIS.

As a graph theoretic method of analysis SSA can be employed in a variety of fields in order to elaborate spatial configurations. Nevertheless, availability of free and open source tools for the graph theoretic analysis of spatial configurations in terms of SSA was limited, which has not only hindered the development of innovation in the respective method of analysis, but also prevented the employment of a proper peer review process for the scientific results reached by employing closed source tools. The basic motivation behind this book has been to overcome these problems by creating a series of SSA Plugins operational on various FOSS4GIS.

The book telling the individual story about how to develop SSA Plugins in FOSS4GIS, on the one hand, has revealed the potential of FOSS4GIS for the development of similar kinds of tools that can be created by employing the scripting facilities in the respective environment, and on the other hand, has uncovered a block box in terms of algorithms and scripts that can be used in
order to create SSA. Because of their flexible and modular architecture allowing to create plugins and scripts via not only IDEs but also console application, FOSS4GIS have a great potential for the analysis of socio-spatial processes. Overall, scripting facilities available in many FOSS4GIS allows us to develop plugins that can perform complex GIS operations. Since the source codes of FOSS4GIS are available for the end users, not only plugins, but also customized applications can be developed by modifying the source code according to the specific needs.
References


Bivand, R.S. and Rundel, C. (2012) rgeos: Interface to Geometry Engine - Open Source (GEOS), R package version 0.2-7, explanations available online at: http://cran.r-project.org/web/packages/rgeos/rgeos.pdf

Plugins and Scripts Developed to Conduct SSA in FOSS4GIS


References


gvSIG (2006b) Scripting guide Version 2, gvSIG 1.0, available on line at: http://downloads.gvSIG.org/download/gvSIG-desktop/dists/1.0.0/docs/gvSIG-1_0-scripting-tutorial-v2-en.zip

131
Plugins and Scripts Developed to Conduct SSA in FOSS4GIS


Michaud, M. (2004) *BeanShellEditor for JUMP (version 0.1.0)*, an installation and user’s guide, available on line at: http://geo.michaelm.free.fr/OpenJUMP/resources/bsheditor4jump-src-0.2.zip

Michaud, M. (2009) *BeanShellEditor for JUMP (version 0.2)*, available on line at: http://geo.michaelm.free.fr/OpenJUMP/resources/bsheditor4jump-0.2.jar


*OpenEV Customization*, available online at: http://openev.sourceforge.net/app/customization.html (accessed 10 July 2012)

Plugins and Scripts Developed to Conduct SSA in FOSS4GIS


Plugin Development with Visual C-sharp, available online at: http://www.mapwindow.org/apps/oldwiki/43.html


Appendices

A. GUIs of the plugin in different FOSS4GIS

**OpenJUMP BeanShell-Jython**

- Space Syntax Analysis
  - Radius for local: 3
  - Options for matrix headings:
    - Use internal index
    - Use a field
  - Choose a field: ineno
  - OK
  - Cancel

**OpenJUMP Java (jar)**

- Space Syntax Analysis
  - Radius for local: 3
  - Options for matrix headings:
    - Use internal index
    - Use a field
  - OK
  - Cancel

**gvSIG Jython GUI**

- Space Syntax A...
  - Radius for local: 3
  - Options for matrix headings:
    - Use internal index
    - Use a field
  - Choose a field: ineno
  - OK
  - Cancel

**gvSIG xml GUI**

- Space Syntax Analysis
  - Radius for local: 3
  - Options for matrix headings:
    - Use internal index
    - Use a field
  - OK
  - Cancel
Appendix

Thuban Python

OpenEV Python

Quantum GIS Python

MapWindow VB.Net
Plugins and Scripts Developed to Conduct SSA in FOSS4GIS

**OrbisGIS BeanShell**

**R Project R Script**

In R Project, no option is actually needed for the creation of an external network data and geodesic matrix as in R you can conduct social network analysis using sna package and calculate many other graph theoretic parameters thanks to the availability of a wide range of packages.

**SAGA GIS C++**
B. GNU General Public License Version 2

Source: http://www.gnu.org/licenses/gpl-2.0.html

GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc.
51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Lesser General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.
For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

**TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION**

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language.
(Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimers of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

   b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

   c) If the modified program normally reads commands interactively when run, you must cause it, when started running for
such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

   a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source distribution of executable or object code) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.
5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the
integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our
decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS
How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

one line to give the program's name and an idea of what it does.

Copyright (C) yyyy name of author

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:
Gnomovision version 69, Copyright (C) year name of author

Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type `show w'. This is free software, and you are welcome to redistribute it under certain conditions; type `show c' for details.

The hypothetical commands `show w' and `show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than `show w' and `show c'; they could even be mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the program `Gnomovision' (which makes passes at compilers) written by James Hacker.

signature of Ty Coon, 1 April 1989
Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License.
Plugins and Scripts Developed to Conduct Space Syntax Analysis in FOSS4GIS: OpenJUMP, gvSIG, OrbisGIS, Quantum GIS, OpenEV, Thuban, MapWindow GIS, SAGA, and R Project

Free and Open Source Software (FOSS) movement has opened the door for Geographic Information Systems (GIS) community to develop and share quality geo-informatics software programs as it was the case for the earlier paths along which the movement made it possible for the other communities to develop quality software programs in different fields. Today, there exists a rich collection of libraries for the development of FOSS for GIS (FOSS4GIS), and they are becoming increasingly available to a larger group of users and developers all around the world.

Parallel to these developments, this book outlines the efforts made for creating plugins for Space Syntax Analysis (SSA) by using FOSS4GIS. On the one hand, it provides a background for those willing to develop graph theoretic plugins by using FOSS4GIS. On the other hand, it develops a concern for the analysis of spatial configurations via SSA by using the plugin designed and created for this purpose in the book.