

An Intelligent Natural-language Interface for ArcView

Jun Xu

Department of Geography, SUNY at Buffalo

junxu@acsu.buffalo.edu

Abstract

The ordinary GIS systems use command line, menu or window interface, which are difficult for a naïve user to use. In this paper, SNePS (the Semantic Network Processing System) is used to represent geographic spatial knowledge. An intelligent natural-language interface between ArcView and SNePS (the Semantic Network Processing System) was implemented. The interface will accept the input command and query in natural language format, transfer it to Avenue command, and then apply it in ArcView.

Key Words: ArcView, SNePS, Natural language

1. Introduction

A main objective of GIS is to allow the user of the system to interact vicariously with actual or possible phenomena of the world (Mark, 1989). The user should be thinking about a problem domain, or a phenomena or a set of phenomena, and not about a computer or program or application (Mark, 1991). A good human-computer interface will help people focus on the work they want to do.

Traditional GIS software, such as ARC/INFO uses command line interpreters. It is difficult for a user to remember so many commands. ArcView and ArcMap use windows interfaces, so people can operate GIS by direct manipulations. Direct manipulation is easier than a command line interpreter, but it is still difficult for a user who is not familiar with GIS. He/She will have to find the tools or buttons that can implement the functions. A natural-language interface is the best choice for naïve users to deal with computers in an easy way. With the advent of real time speech-understanding systems, spoken natural language is likely to become a very important form of system interaction for GIS (Mark, 1991). It will be useful for both naïve users and GIS experts.

A knowledge base is needed to build the natural-language interface. The fundamental issue in developing a knowledge base system is knowledge representation. The knowledge representation system should not only represent the input knowledge in the knowledge base, but also infer unknown information from the knowledge it has been told. In this paper, SNePS (the Semantic Network Processing System) is used to represent geographic spatial knowledge. An intelligent natural-language interface between ArcView, a geographic information management system, and SneBR, the SNePS Belief Revision system, is implemented. The interface accepts input commands and queries in natural-language format, transfers it to Avenue commands, and then applies it in ArcView. The interface can also do the work of spatial reasoning.

2. Previous work

Representing knowledge in natural language is a very important issue in building a natural-language interface. Haller and Mark (1990) addressed the problem of generating natural language to express spatial relations between or among geographic entities. Multiple representations of objects are needed in geographic information system. For example, a city could be a point or an area on the map, depending on the scale of the map. SNePS was used to multiply represent locative expressions. Since SNePS is an intensional knowledge representation system, it supports multiple representation of what could be one physical object (Shapiro and Rapaport, 1987). Zhan and Mark (1992) described an object-oriented spatial knowledge representation schema in CLIPS/COOL environment for representing and processing spatial knowledge.

Current GISs primarily use quantitative models and representations, while natural language spatial relations are usually dominated by qualitative properties. To formalize people's use of spatial relations in natural language, Egenhofer and Shariff (1998) developed a formal model that captures *metricdetails* for description of natural-language spatial relations so that they can be represented within GIS. The model of topological relations with metric refinements was calibrated for a set of 59 English-language spatial terms, for which 34 human subjects had generated sketches. This model enables the generation of simple sentences to describe spatial scenes and processing of spatial queries with natural-language spatial term (Shariff, et al. 1998). But they did not discuss how to implement this model in GIS.

An intelligent natural-language user interface has been implemented by connecting ARC/INFO with SNACTor, the SNePS acting component (Shapiro, et al. 1992). The

communication medium between ARC/INFO and SNePS is the file system. Commands from SNACTOR to ARC/INFO are issued by generating command files; result from command executions in ARC/INFO are brought back into SNACTOR via watch files. The SNACTOR side of the communication is defined in a Common Lisp file. Its function is to translate the request into a set of AML commands, and write them to a command file. Then SNACTOR will wait for ARC/INFO to implement the commands. The ARC/INFO side of the communication makes use of AML. ARC/INFO look into the interface directory, if the command file exists, it executes it, then delete the command file and write the result to the watch file. Once the command file is deleted, SNACTOR adds the result of the command to its knowledge base, and then is ready to accept the next natural-language requirement.

Another intelligent GIS interface was designed by Emergency management team in PSU (Rauschert, et al. 2002). They developed a Dialogue-Assisted Visual Environment for GeoInformation (DAVE_G), which uses a multimodal, multi-user GIS interface that can access GIS data through voice and gesture. It is based on a multimodal interface framework and applies ArcObjects for providing necessary GIS functionalities. The hand trajectory captured by a camera is used to recognize gestures and supports the interpretation of a user's spoken command. The recorded human voice is processed by standard speech recognition software. The speech and hand gesture are integrated to generate a command. Although natural language is used in this interface, it is translated into GIS commands by speech recognition software. There is no knowledge base to represent information, and there is no reasoning of knowledge. The result of implement the command is displayed, and there is not feedback information from the GIS system.

3. Semantic network knowledge representation

3.1 Introduction to SNePS

SNePS, the Semantic Network Processing System, is an intensional, propositional, semantic-network knowledge representation system that is used for research in artificial intelligence and in cognitive science (Shapiro and Rapaport, 1995). A SNePS-based knowledge base could be built by a human informing it using a natural language. It uses an intensional propositional semantic network to represent knowledge, and supports multiple representation of what could be one physical object. So it should be able to represent anything and everything expressible in natural-language, it should be able to represent generic, as well as specific information; it should be able to use generic and specific information to reason and infer information implied by what it has

been told (Shapiro and Rapaport, 1992). SNePS contains SNeBR, the SNePS Belief Revision system. Therefore there is always a “current context” specified, which consists of a set of hypotheses asserted to the system by the user, and a “current belief space” consisting of the current context and all propositions so far derived from them (Shapiro and Rapaport, 1992).

SNePS is a programming language whose primary data structure is a semantic network. A Semantic network is a labeled, directed graph whose nodes represent entities and whose arcs represent binary relations between entities (Shapiro and Rapaport, 1995). It is a propositional semantic network, which means that all information, including propositions, “facts”, etc, is represented by nodes. Base nodes are distinguished by having no arcs coming from them. A base node is assumed to represent some entity—individual, object, class, property, etc. For example, there are two base nodes in Figure 1, “Buffalo” and “city”. No two nodes represent the same, identical entity. Molecular nodes may represent propositions. In Figure1, M1 is a molecular node. Arcs represent the relations between nodes. There are two arcs in figure1, one is labeled with “member” and points to “Buffalo”, and the other one is labeled with “class” and points to “city”. The network in figure1 represents that Buffalo is a city.

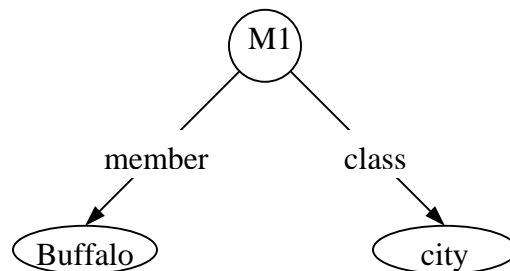


Figure1. A semantic network representation for Buffalo is a City

3.1 Geographical knowledge representation and reasoning in SNePS

In GIS, we have to express geographic locations of objects. A molecular node with three arcs

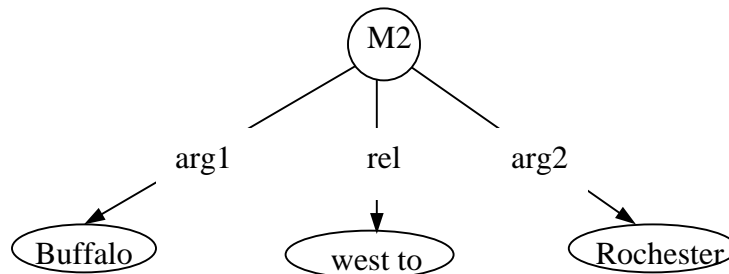


Figure2. A representation of geographic locative

is used to represent the geographic locative information. Two nodes pointed by “arg1” and “arg2” are two geographic objects, and “rel” is the relation between two objects. Figure 2 is the representation of “Buffalo is west to Rochester”.

SNePS can not only represent information, but also infer information from what it has been told. To do inference, a knowledge base has to be built. Figure 3 is a simple map of New York State. We use SNePS to represent some basic information on the map, and infer information that is not told. The semantic network in Figure 4 can represent all the information we tell SNePS and inferred by SNePS.

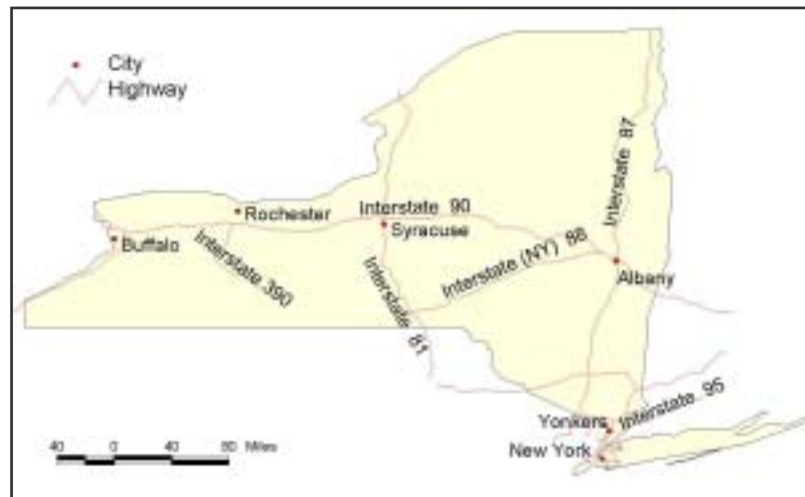


Figure3. A simple map of New York State

The following is the annotated sample run in SNePS. The lines begin with “:” is what the user input to SNePS, the following line is the response of SNePS. At first, tell SNePS some basic information.

```
: Buffalo is a city.  
I understand that Buffalo is a city.  
: Rochester is a city.  
I understand that Rochester is a city.  
: Buffalo is west to Rochester.  
I understand that Buffalo is west to Rochester.
```

The propositions that Buffalo is a city and Rochester is a city are represented as M1 and M2 respectively in Figure 4. The molecular node M3 represents the proposition that Buffalo is west to Rochester. Now if we ask SNePS whether Rochester is east to Buffalo, SNePS does not know the answer because there is no such information in the network.

: Is Rochester east to Buffalo?
 I really don't know if Rochester is east to Buffalo.

We have to tell SNePS that if a city is west to another city then the second city is east to the first city.

: If a city is west to an object then the object is east to the city.
 I understand that if a city is west to a object then the object is east to the city.

We ask SNePS the same question again. SNePS will find that Buffalo is west to Rochester in the network. And form the rule we have just told it, it can refer that Rochester is east to Buffalo. The molecular node M4 in Figure 4 is built in the network automatically. This time SNePS gives the positive answer.

: Is Rochester east to Buffalo?
 Yes, Rochester is east to Buffalo.

Now we tell SNePS more information. The proposition that I-90 is a highway is represented by node M5, and the proposition that Buffalo is on I-90 is represented by node M6 in Figure4.

: I-90 is a highway.
 I understand that I-90 is a highway.
 : Buffalo is on I-90.
 I understand that Buffalo is on I-90.

If we ask whether Buffalo is on highway, SNePS will find from the network that Buffalo is on I-90, and I-90 is a highway, so it will give the positive answer.

: Is Buffalo on highway?
 Yes, Buffalo is on I-90.

Figure 4 shows the whole semantic network that has just been built.

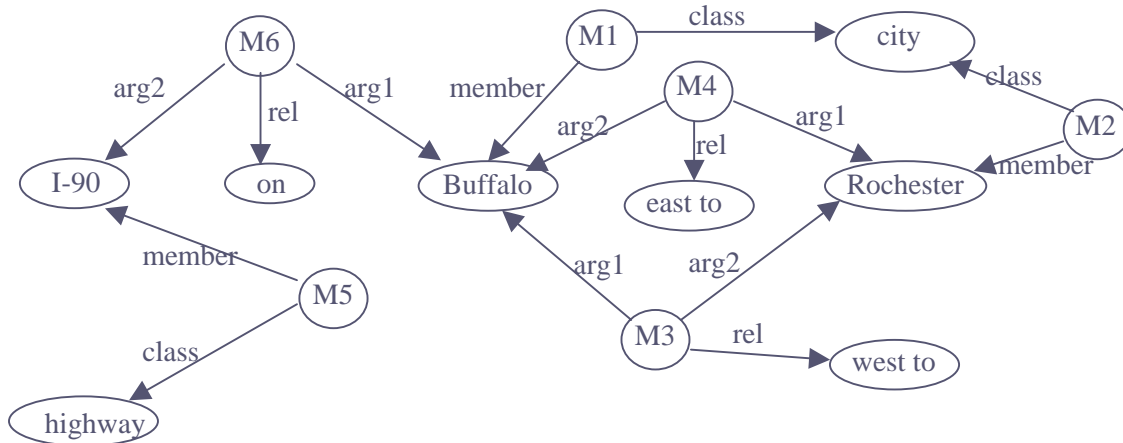


Figure4. An example of semantic network

4. Interface between ArcView and SNePS

Unlike ARC/INFO system, ArcView uses windows interface. Many functions are implemented with menus, buttons, or tools. All this menus, buttons and tools are associated with some embedded scripts that are written in Avenue. With Avenue, the user can also integrate ArcView with other applications. To operate an ArcView project in natural language, we need to implement an interface that integrates ArcView with SNePS.

In this paper, the same protocol used by Shapiro et al (1992) in the ARC/INFO SNACTOR system is used. The communication medium is also the file system. SNePS issues a command, generates a command file, and waits for a result to be generated. ArcView reads the command file, executes the command, generates a result, writes the result to a result file if needed, and waits for the next command. There are two sides of the project, ArcView side and SNePS side.

4.1 ArcView side

In ArcView side, the interface receives Avenue command from SNePS, and runs it. All the Avenue files are restored as scripts in ArcView projects. The main script is *MyStartupScript* which is written in Avenue:

```
MsgBox.Info("Please input command in Sneps.", "To start...")
f = "$Home/arcview/interface/next.cmd".asFileName
File.Delete(f)
_continue = true

while (_continue)
  if (File.Exists(f)) then
    cmdFile = TextFile.Make(F, #FILE_PERM_READ)
    cmd = cmdFile.Read(cmdFile.GetSize)
    curScript = Script.Make(cmd)
    If (curScript.HasError.Not) then
      curScript.DoIt("")
    end
    cmdFile.Close
    File.Delete(f)
  end
end

av.GetProject.close
av.Quit
```

MyStartupScript is set as default start up script of the ArcView project. Whenever ArcView is started, *MyStartUpScript* will be running, and performing a command loop as long as the global variable *_continue* is TRUE. When it detects the existence of file *next.cmd*, it will read it, and execute the Avenue command in the file. The sentence "*curScript.DoIt(*''*)*" in *MyStartupScript* will execute the commands in the command file. Then command file is deleted.

The script will go on doing this loop till *_continue* is set to FALSE. It is done by “*stop*” action in SNePS.

4.2 SNePS side

In SNePS side, it gets input information, parses it, and performs the corresponding action function. The functions of primitive action are defined in file *arcview.lisp*. These primitive actions transfer the actions need to be performed to the form of Avenue commands. For example, to turn on a theme, the action “*turnon*” is defined:

```
(define-primaction turnon (object1)
  (setf thisobject (coerce-node (lex-node object1) :string))
  (setf cmd (format nil "av.run(\"TurnonTheme\", \"~a\")" thisobject))
  (just-do-it (format nil "Turn on theme ~a" thisobject) cmd))
```

In this action, function *lex-node* return a node at the head of the *lex* arc from *object1*, and function *coerce-node* changes node to string. The action “*turnon*” formats a Avenue command “*ac.run(“TurnonTheme”, “aTheme”)*”, and calls “*just-do-It*” to execute this command. This command will run an embedded ArcView script *TurnonThemes*. Function “*just-do-It*” is defined in ARC/INFO SNACTOR (Shapiro, et al. 1992).

```
;This is from ARC/INFO SNACTOR demo
(defun just-do-it (command command-string &optional (print-result nil))
  (format t "~&~% Now doing: ~a~%" command)
  (execute-command command-string)
  (when print-result (print-result)))
```

Function “*just-do-It*” calls function “*execute-command*” to execute the command. Function “*execute-command*” is also defined in ARC/INFO SNACTOR (Shapiro, et al. 1992). It writes the Avenue command to a command file named *next.cmd*.

```
; This is from ARC/INFO SNACTOR demo
(defun execute-command (command-string)
  "Takes a COMMAND-STRING and writes a file next.cmd into the communication
interface directory. The command loop running in the ArcView process waits
for this file, executes it and deletes it."
  (wait-for-completion)
  (with-open-file (nextcom *nextcom* :direction :output)
    ;; next.cmd takes as an argument the name of the communication
    ;; directory viewed from the ArcView process
    (format nextcom "~a~%" command-string))
  (wait-for-completion))
```

When the user wants to stop Arcview, a stop action is performed. Stop action is defined as following:

```
(define-primaction stop ()  
  (just-do-it "quit" "_continue = false"))
```

This will write to *next.cmd* an Avenue sentence “_continue = false”, which set the Avenue global variable “_continue” to false.

4.3 Connection between SNePS and ArcView

The connection of SNePS and ArcView happens in the *interface* direction. Every time SNePS get a command, it transfers it into Avenue command, and saves it in a command file *next.cmd* in *interface* direction. Usually the commands in *next.cmd* are Avenue commands which call an embedded ArcView script. After *MyStartupScript* in ArcView detects the existence of *next.cmd*, it will execute this command, and run the corresponding script. For example, if the command in *next.cmd* is “*ac.run(“TurnonTheme”, “aTheme”)*”, the script *TurnonTheme* is called by the sentence “*curScript.DoIt(“”)*” in *MyStartupScript*. Script *TurnonTheme* sets the theme to be visible if it is not:

```
theview = av.GetActiveDoc  
theTheme = theView.FindTheme(self) `self get the value of aTheme  
if (theTheme.IsVisible.not) then  
  theTheme.SetVisible(TRUE)  
end
```

Because the action “turn on a theme” will display the result in the window, so we don’t need a result file from ArcView. For some actions, such as “*find*” need to query on the map and get result from the map, the result will be written to a result file named *comout*. SNePS reads the result from the result file and prints it out.

4.4 Sample run

This is an annotated sample run of this project. This demo is run on a SUN workstation in Geographic Information and Analysis Laboratory (GIAL) at Department of Geography. The output result has been edited for saving space. User input in SNePS package is shown at “*” prompts, user input in SNePSUL package is shown at “-->” prompts, and user input in natural-language is shown at “:” prompts. The system’s response follows after the user input separated by a blank line.

At the beginning of the demo, reset the network, and define relations and path of the network.

```

* (demo "ucgis.sneps")

File /home/grad/geo/junxu/arcview/ucgis.sneps is now the source of input.

* (resetnet t)

Net reset

;;; Arc labels for predicates and relations
(= (define member class sub sup lex object property rel arg1 arg2
    object3) NEW-RELATIONS)

(MEMBER CLASS SUB SUP LEX OBJECT PROPERTY REL ARG1 ARG2 OBJECT3)

* (define-path class (compose class (kstar (compose member- class)) (kstar
(compose sub- sup))))
CLASS implied by the path (COMPOSE CLASS (KSTAR (COMPOSE MEMBER- CLASS))
(KSTAR (COMPOSE SUB- SUP)))
CLASS- implied by the path (COMPOSE (KSTAR (COMPOSE SUP- SUB))
(KSTAR (COMPOSE CLASS- MEMBER)) CLASS-)

```

Then go to SNePSUL package and read in the grammar and the lexicon files. The lexicon file defines the words and phases will be used in operating ArcView.

```

* ^^
--> (lexin "av-lex.dat")
undefined- (NIL NIL NIL NIL NIL NIL NIL)
("view" "theme" ..... "to" "of")
--> (atnin "av-grammar.dat")
State S processed.
State RESPOND1 processed.
.....
State GTHRESH processed.
State END processed.

Atnin read in states: (END GTHRESH ..... S NIL)

```

Now set some variables that control input reading and tracing, and attach the primitive functions with LISP functions.

```

--> (setq \*terminating-punctuation-flag\* '("." "!" "?")
      \*infertrace\* nil
      parser::\*debug\* nil)
NIL
--> (attach-primaction
    ;; built-in actions:
    (build lex "snsequence") snsequence
    .....
    (build lex "stop") stop)
T

```

Ready to parse the input natural language now. Go to the natural-language input and parse loop.

```

--> (parse -1)

ATN parser initialization...

Input sentences in normal English orthographic convention.
Sentences may go beyond a line by having a space followed by a <CR>

```

To exit the parser, write ^end.

Now we can parse the input command, and operate on ArcView. At first, we have to define some classes, such as view, document, and theme, the relations between classes, and the rules on operating them in ArcView. To save space, this part is not shown here.

Then we can ask SNePS to perform something on ArcView. The results in ArcView are shown in Figure 5. First, ask it to open a view. After opening a view, SNePS will believe that view1 is opened, and is active.

```
: Open view1.
```

```
I understand that you want me to perform open on view1.  
Now doing: open view1
```

View1 is opened in ArcView (Figure 5.a). Then add a theme to view1. Since perform “adding theme to a view” has a precondition that the view must be active, and it is satisfied, SNePS adds *statetheme* to view1. The New York State map will be displayed in the view (Figure 5.b). The SNePS will do the same thing for adding *roadtheme* and *citytheme*.

```
: Add statetheme to view1.
```

```
I understand that you want me to perform add on statetheme and view1.  
Now doing: Add theme statetheme to view1.
```

```
: Add roadtheme to view1.
```

```
I understand that you want me to perform add on roadtheme and view1.  
Now doing: Add theme roadtheme to view1.
```

```
: Add citytheme to view1.
```

```
I understand that you want me to perform add on citytheme and view1.  
Now doing: Add theme citytheme to view1.
```

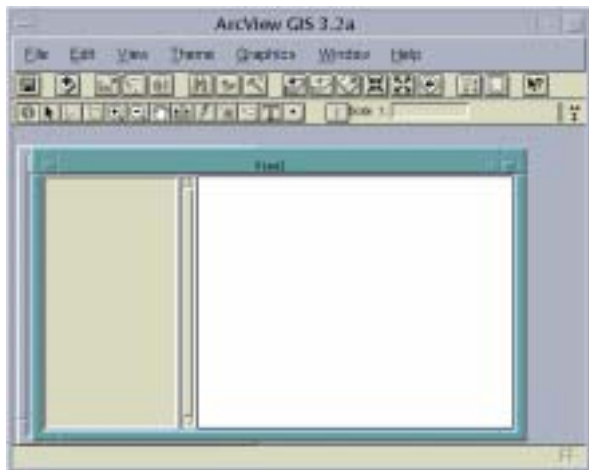
Three themes have been added to view1 (Figure 5.c).

We can query on ArcView using SNePS. For example, find Buffalo in *citytheme*. SNePS can also get feedback from SNePS how many features are selected in ArcView. The selected feature is highlighted in ArcView (Figure 5.d).

```
: Find Buffalo in citytheme.
```

```
I understand that you want me to perform find on Buffalo and citytheme.  
Now doing: query Buffalo in citytheme.
```

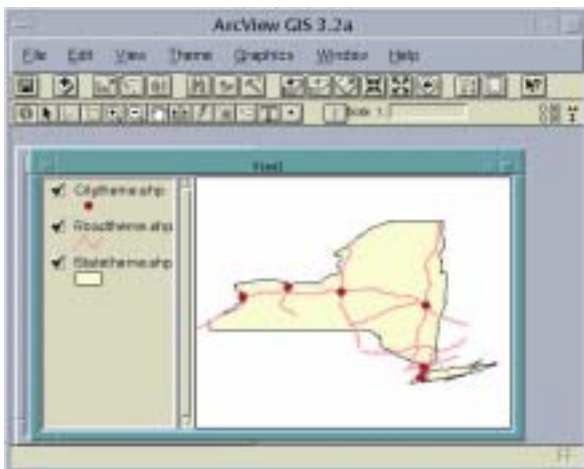
```
There are 1 features selected.
```



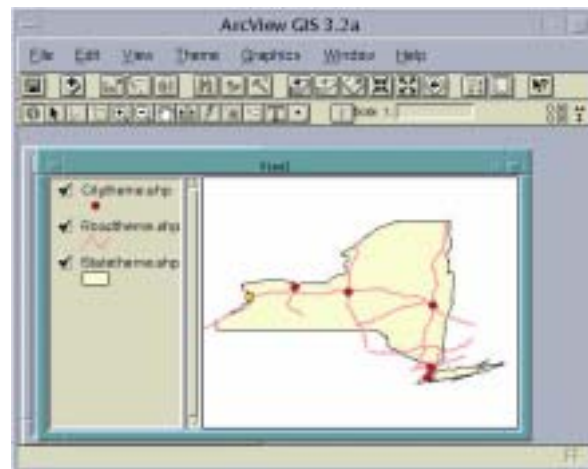
a



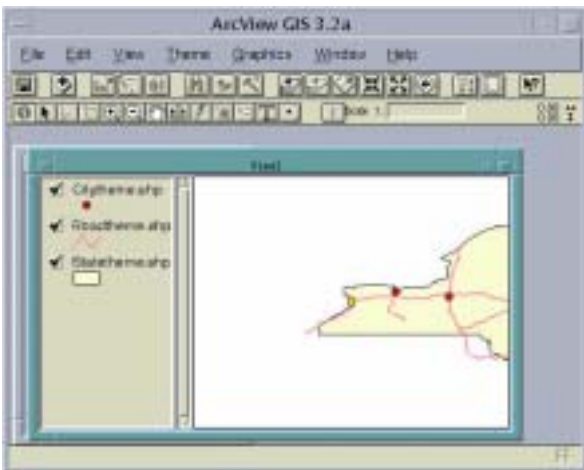
b



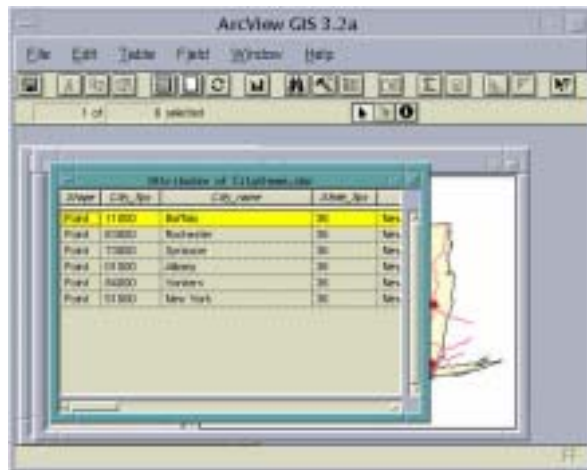
c



d



e



f

Figure5. Illustration of sample run in ArcView

Zoom in a theme will zoom to the selected features in the theme in an active view. If we ask SNePS to perform zooming in *citytheme*, ArcView will zoom to Buffalo in view1 (Figure 5.e).

Zoom out will zoom to the full extent in the view.

```
: Zoom in citytheme.
```

```
I understand that you want me to perform zoom in on citytheme.  
Now doing: Zoom to selected in citytheme.
```

```
: Zoom out statetheme.
```

```
I understand that you want me to perform zoom out on statetheme.  
Now doing: Zoom to full extent.
```

Ask SNePS to show the table of *citytheme*. The attribute table of *citytheme* will be displayed in ArcView, and the record of the selected city is promoted to the top of the table (Figure 5.f).

```
: Show table of citytheme.
```

```
I understand that you want me to perform show on table and citytheme.  
Now doing: Show the table of citytheme.
```

Perform stop will quit ArcView.

```
: stop arcview.
```

```
I understand that you want me to perform stop on arcview.  
Now doing: quit
```

```
: End of /home/grad/geo/junxu/arcview/ucgis.sneps demonstration.
```

5. Compared with previous work

This project uses the same structure and protocol as the previous ARC/INFO SNACTOR project. But the method to implement the protocol is a little different. Since ARC/INFO uses command line, and its Macro language AML allows the user to write program containing arbitrary ARC/INFO commands, SNePS only need to translate input language to ARC/INFO commands, and write them to command file. But there is no language can operate on ArcView directly. The ArcView Avenue language must be written in a script file in an ArcView project, and usually a set of commands is needed to apply one function. So I have to write those sets of Avenue commands which apply different ArcView functions in different script files. The sentences in the command file are only commands to run those script files.

The previous ARC/INFO SNACTOR project only implements a few ARC/INFO operations, and can only get simple query result from ARC/INFO. This is because of the limit of ARC/INFO itself. ARC/INFO can only query on one map layer, whereas ArcView can display multi-layer of

maps in one frame, and query with more than one map. Although this demo has not applied this complicate query in natural language at present, it is possible.

The different of this project with DAVE_G is that this project use SNePS to represent the knowledge, so it can not only operate on the GIS system, but also reason and remember the represented knowledge. While DAVE_G is only a interact environment that the user can operate on GIS system, like any other existed interface, except it is more human friendly.

6. Conclusion and future work

An intelligent natural-language interface of ArcView is described in this paper, and the semantic network is used to represent the geographic knowledge. The interface accepts natural-language commands with SNePS, transfer the commands to AVENUE commands, then operate on ArcView, and get the response.

Currently the interface can only perform the basic query on one single map and one single field in the attribute table. It cannot do complex query such as “Find all the cities have more than 1,000,000 populations”. It cannot query from two maps either, such as “Find all the cities in the distance of 5 miles to Intersate 90”. The present grammar file is not enough to parse such complicate sentences. The future work to improve the query function needs to verify the grammar file. The multiple representation of the equivalent objects is another issue to be done in the future. For example, if we can represent the proposition such as “my favorite city” and “near”, “far”, we can query questions such as “find my favorite city in city map”, “find cities near Buffalo”.

In this project and previous ARC/INFO SNACTor project, only the input knowledge is represented in the network, the knowledge required from the query is not represented very well. We get “New York City” is one of the cities having more than 1,000,000 population. If we can build this knowledge into the network, we can use it late when someone asks “Does New York City has more than 1,000,000 populations?” So that it becomes a real intelligent interface.

Reference

Egenhofer, M.J., and Shariff, A. R., 1998. Metric Details for Natural-Language Spatial Relations. *ACM Transactions on Information Systems*, 16 (4): 295-321.

- Haller, S. M., and Mark, D., 1990. Knowledge representation for understanding geographic locatives. Proceedings, 4th International Symposium on Spatial Data handling, July, 1990, v.1, 465-477.
- Mark, D., 1989. Cognitive Image-Schemata and Geographic Information: Relation to User View and GIS Interface. Proceedings, GIS/LIS'89, Orlando, 551-560.
- Mark, D., 1991. User interfaces for geographic information systems: Toward a research agenda. Proceedings of the Eleventh Annual ESRI User Conference, v. 2, pp.525-530.
- Rauschert, I., Agrawal, P., Fuhrmann, S., Brewer, I., Wang, H., Sharma, R., Cai, G., & MacEachren, A, 2002. Designing a Human-Centered, Multimodal GIS Interface to Support Emergency Management. ACM GIS'02, 10th ACM Symposium on Advances in Geographic Information Systems, Washington, DC, USA, November, 2002.
- Shapiro, S.C., Chalupsky, H., Chou H., and Mark, D., 1992. Intelligent user interfaces: Connecting ARC/INFO and SNACTor, a semantic network based system for planning actions. In *Proceedings of the Twelfth Annual ESRI User Conference, V. 3*, pages 151-165. Environmental Systems Research Institute, Redlands, California, 1992.
- Shapiro, S.C., and Rapaport, W.J., 1987. SNePS considered as a fully intensional propositional semantic network. In N. Cercone and G. McCalla (eds) *The Knowledge Frontier, Essays in the Representation of Knowledge*, Springer-Verlag, New York, 363-315.
- Shapiro, S.C., and Rapaport, W.J., 1992. The SNePS family. *Computer for Mathematic application*, 23(2-5): 243-275.
- Shapiro, Stuart C. and Rapaport, William J. (1995), "An Introduction to a Computational Reader of Narrative", in Judith Felson Duchan, Gail A. Bruder, & Lynne E. Hewitt (eds.), *Deixis in Narrative: A Cognitive Science Perspective* (Hillsdale, NJ: Lawrence Erlbaum Associates): 79-105.
- Shariff, A.R., Egenhofer, M.J., and Mark, D.M., 1998. Natural-Language Spatial Relations Between Linear and Areal Objects: The Topology and Metric of English-Language Terms. *International Journal of Geographical Information Science*, 12 (3): 215-246.
- Zhan, F., Mark, D., 1992. Object-oriented spatial knowledge representation and processing: formalization of core classes and their relationships. Proceedings, Fifth International Symposium on Spatial Data Handling, Charleston, South Carolina, v.2, 662-671.