Abductive Spatial Reasoning Diagnosis Models

Goal

The goal of this project was to research causal reasoning models for medical diagnoses that incorporate spatial reasoning and hierarchical knowledge. In the past, many studies have demonstrated that doctors make avoidable errors such as incorrect diagnoses, and that these might be preventable if computer decision aids were available to physicians. One critical tool that humans and computers use is called abductive reasoning [J1, P1, R1, T1]. In many fields, such as medical diagnosis or criminal investigation, people are faced with evidence about a situation and must come up with a plausible explanation for that evidence. For example, a doctor who learns of a patient's symptoms (the evidence) must come up with a diagnosis (the explanation) for what is wrong. A critical aspect of such reasoning involves the use of cause-effect knowledge to generate and evaluate plausible explanatory hypotheses. However, computational modeling of human abductive reasoning has proven to be very challenging. At present, general-purpose abductive reasoning models that are both effective and robust in handling real-world applications simply do not exist. With this project, I investigated new methods for automated abductive reasoning and applied them to diagnosing focal brain damage.

Background

Substantial past effort has been made to represent causal knowledge in a form that computers can interpret and to capture abductive reasoning processes in computational models [B1, N1, P2, T2]. This past work has been motivated both by a desire to better understand human cognitive processes (psychology) and to provide computer decision-support software that can improve human decision-making (AI). Past computational abductive models typically construct preferred hypotheses that not only explain the observed data features, but do so while being as parsimonious/simple as possible (Occam's Razor). For example, consider a toxicologist analyzing a chemical spill. If certain symptoms appear in the water (e.g. radioactivity, metals, an increased gravity), then that allows the toxicologist to hypothesize a set of possible solutions. Unlike with object hierarchies (classifying one object as part of a larger entity altogether), only relatively limited past work has investigated the representation of and reasoning with hierarchies of causal relations. One of the key goals with this project will be to extend hierarchical representation and reasoning with multi-level causal knowledge, and compare it with past abductive algorithms and bottom-up methods for hypothesis construction, such as swarm intelligence [L1] and self-assembly [G1].

Question

If abductive reasoning is to work in real-world applications, one needs a way to encode spatial information. This problem is the overarching issue with this project. The question I sought to answer was: *how should spatial information best be represented and applied to focus hypothesis formation and evaluation*? I used diagnosing focal brain damage as a case study. Because there are very clear definitions of what constitutes brain damage, the output of the expert system can be easily compared to previous diagnoses, and thereby avoid ambiguity.

Results

From this research, I accomplished three objectives. First, I designed a knowledge representation language for causal relations that incorporated spatial relations, hierarchical informatics, and variables. This language can be written into a single text file, forming knowledge base, which can then be parsed by a computer. The language is divided into four sections: parameters, functions, objects, and attributes.

Objects are like nouns. In the language, they provide information about some kind of physical entity. These entities can have specific properties such as synonyms, locations, and parts. Parts are especially important in order to develop spatial relations. If an object has a list of parts, it means that all the entries in that list form the object's composition. Consider the following example: suppose we declare an object called building. Its parts might include pipes, electrical wiring, floors, windows, and ceilings. Likewise, a window might be an object of its own with its parts including glass, frame, and blinds. Besides having parts, objects can also have a property called location. Location is another way of showing spatial relationships. Instead of listing the components of an object, location tells where the object would appear on an XYZ axis. Using these tools, one can write a language that describes a very complex object by breaking it down into smaller components and showing the spatial relations.

Although an object is an important tool for establishing causal relations, it is not the only one. Attributes are another important device. They can almost be thought of like variables, albeit ones where only specific values defined in the knowledge base are allowed. The user can determine the values for some of the attributes- these ones are called input attributes. Possible input attributes- using earlier chemical spill example- might include radioactivity, color of the water, and presence of metals. The two most common properties an attribute can have are synonyms and values. The synonyms are simply alternative names for the same attribute. The values are the list of valid statuses an attribute can have. Values are capable of having nested properties of its own, which help to create hierarchical information such as elaborations and descriptions. Elaborations, in concept, are similar to an object's parts in that they provide further information about a value. A description, as the name suggests, offers an explanation for the value. Continuing with the chemical spill example, suppose that radioactivity was an attribute. It is possible values could be "Yes" and "No". If there it turns out the water is radioactive, one could have an elaboration for the "Yes" value. The possible values for the elaboration could be the type of radiation (alpha, beta, and gamma).

In order to complete the task of diagnosis, another concept is needed: inferred attributes. Inferred attributes are possible explanations for the observed maladies. Each inferred attribute has a list of possible symptoms (called effects); these are the causal relations. As a computer continues through a diagnosis session, and it collects more input attributes, it evaluates the possibility of one (or several) maladies being the answer to the observed ailments.

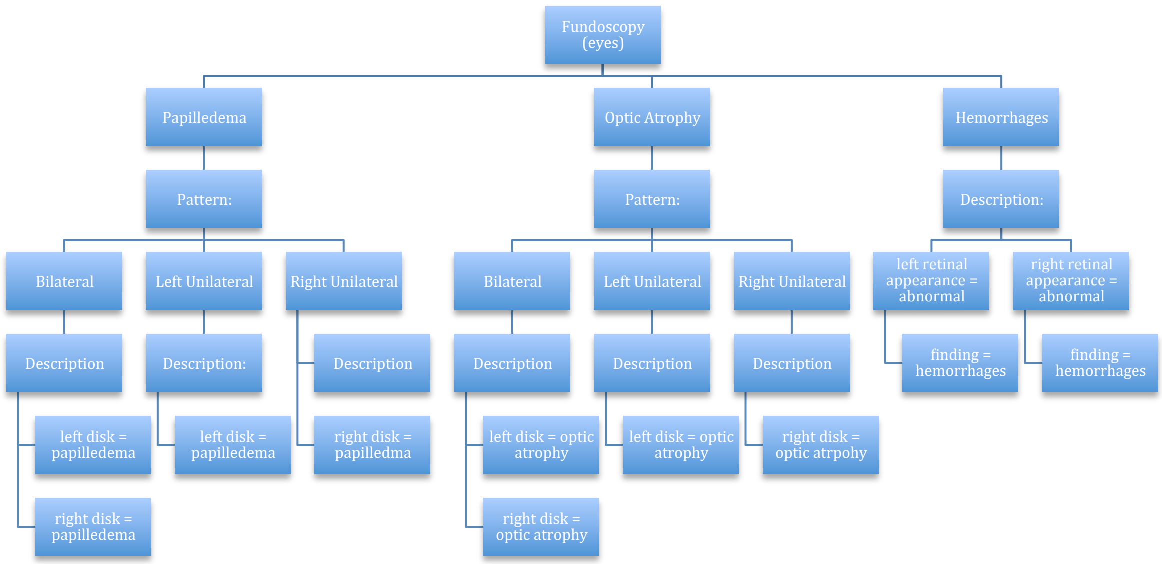
Parameters are lists of adjectives. After being declared at the top of the knowledge base, they never appear by themselves. They are always used to modify an object or attribute's name or entry in one of their properties. Suppose that there are several objects that have the exact same properties, and even similar sounding names. With parameters, just one object can be declared, but will have a parameter marked by brackets. Thus, when a computer reads the object, it will create one object for each entry in that particular parameter's list. An example a parameter in the current version of the knowledge base is called side. It has two values called "left" and "right". After declaring this attribute, left and right will replace the word <side> wherever it appears in the knowledge base.

Functions are related to parameters. In the language, a function can take in some variable (representing a parameter), searches through a predetermined list of valid inputs, and then returns the corresponding output. In the current version of the knowledge base for this project, there is only one function called "contralat". It takes in a variable called s, which is assumed to represent a value from the parameter called side- meaning that s can either be left or right. If s turns out to be left, the function will return right. And if the s is right, the function will return left. Using these four elements of the knowledge language (objects, attributes, parameters, and function) someone can write a knowledge base encoding information about any topic, ranging from plumbing, chemical spills, or focal brain damage.

The second accomplishment was implementing a preliminary parser capable of reading in a text file containing a knowledge base. Using MATLAB, the parser does two things. First, it checks that all of the syntax rules for the language are followed. If it finds any problems, such as a missing parenthesis or semicolon, it will print out an error message and the line where the problem occurred. This is a basic function of any parser. However, the parser developed for this project is also capable of creating data structures that house all of the knowledge in a form that a computer can readily access. Once the parser has completed its job and assuming there were no errors, the parser will save all of the information into .mat files that other MATLAB programs can access and do computations with.

The third accomplishment was applying the knowledge language to neurological localization of brain damage. In this knowledge base, the objects represent the human anatomy. The objects include parts of the anatomy such as the head, arms, and legs. However, most of the objects detail with the human brain. For the objects that are part of the human brain, the XYZ coordinates are used in addition to parts lists. The units used in location roughly correspond to centimeters, but the measurements are not precise. The numbers are sufficient to illustrate where the different parts of the brain are in relation to each other, which is the one of the goals of having objects in the first place.

The input attributes cover a wide range of medical examinations. Among the input attributes are respiration, eye exam, level of consciousness, and plantar reflexes. An excellent example that illustrates the hierarchy of attributes is the one called "fundoscopy(eyes)". There are three possible values for the attribute: papilledema, optic atrophy, and hemorrhages. For the first two attributes, they have further elaborations that ask questions about the patterns the eyes have. The possible values for the attributes are bilateral, left unilateral, and right unilateral. After these levels, there is a description for each value.



(This figure illustrates the hierarchy for the attribute fundoscopy(eyes) and all of its possible values, elaborations, and descriptions)

As stated earlier, these input attributes have causal relations with inferred attributes. In the current version of the knowledge base, the inferred attributes (therefore, the possible answers) are: increased intracranial pressure, herniation syndrome, left supratentorial lesion, right supratentorial lesion, subtentorial lesion, diffuse encephalopathy, and brain death.

To fully understand how this knowledge base could be used to diagnose a patient, consider the following example. The complete coma knowledge base is written into a text file and parsed using the MATLAB program called parser. This loads the data structures and all of the data written into the knowledge base. During the questioning session, the diagnostician answers questions that tell the following about the patient:

The patient is in a stupor.

The patient is hyperventilating.

The patient’s pupillary responses are bilateral, dilated, and fixed.

The resting position of the patient’s eyes is dysconjugate.

The spontaneous movement of the patient’s eyes is ocular bobbing.

The patient’s eye response to head turning is asymmetric

Ice water calorics are absent.

When the patient grimaces, there is unilateral weakness.

The patient's limbs are symmetrical.

The patient's motor response to pain shows hemiparesis.

The patient's muscle stretch reflexes exhibit bilateral hyperreflexia.

The patient's plantar reflexes are extensor.

With attributes set to these values, the computer would evaluate the probabilities of each of the values and look for the simplest explanation that explains all of these input attributes. It might take several inferred attributes to diagnose these conditions. But in this case, the computer would come up with only one diagnosis: subtentorial lesion.

Conclusion

After this research project, I can conclude the following:

The knowledge base language is sufficiently developed to integrate and support hierarchical causal knowledge with anatomical information. As shown with the fundoscopy example, hierarchy of attributes can be used to represent data regarding the human anatomy. Should modifications be needed for the knowledge base, it would be a simple matter to modify the hierarchy to reflect that. Additionally, the existence of the objects (and especially their spatial relationships) shows where any given attribute would fit in with the rest of the body.

The apparatuses used to represent spatial information can all be supported. As shown with the information about the spatial relations, one can use both the list of parts as well as the XYZ plane to efficiently describe where any given object is located. It does not require a lot of writing, just a line or two in the text file containing the knowledge base.

The knowledge base can be cleanly parsed and wrapped into internal data structure successfully. The parser written in MATLAB can read in a file containing a knowledge base, and create data structures containing all of the information in the file. From here, other programs (such as a diagram using abductive reasoning) can use the information now stored in .mat files.

With these successes, it is possible to use a language to create a knowledge base about focal brain damage, parse it into MATLAB, and then use those resulting data structures. The next steps are to develop a decision support system that makes use of these data structures, and will actually perform abductive reasoning to diagnose focal brain injuries.

References

[B1] Bylander T, Allemang D, Tanner M, Josephson J. The Computational Complexity of Abduction, *Artificial Intelligence,* 49, 1991, 25-60.

[G1] Grushin A, Reggia J. Automated Design of Distributed Control Rules for Self-Assembly of Pre-specified Artificial Structures, *Robotics and Autonomous Systems,* 56, 2008, 334-359.

[J1] Josephson J, Josephson S. *Abductive Inference,* Cambridge University Press, 1994.

[L1] Lapizco-Encinas G, Reggia J. Diagnostic Problem Solving Using Swarm Intelligence, *Proc. IEEE Swarm Intelligence Symposium,* 2005, 365-372.

[N1] Neapolitan R. *Learning Bayesian Networks,* Prentice Hall, 2004.

[P1] Patokorpi E. What Could Abductive Reasoning Contribute to Human Computer Interaction? *PsychNology Journal,* 7, 2009, 113-131.

[P2] Peng Y, Reggia J: *Abductive Inference Methods for Diagnostic Problem-Solving,* Springer, 1990.

[R1] Reggia J: Abduction, *Encyclopedia of Artificial Intelligence,* S. Shapiro (Editor in Chief), Wiley-Interscience, 1992, 2-3.

[T1] Thagard P, Shelley C. Abductive Reasoning: Logic, Visual Thinking, and Coherence, Philosophy Dept. Technical Report, University of Waterloo, 1997.

[T2] Thomas R, Dougherty M, Sprenger A, Harbison J. Diagnostic Hypothesis Generation and Human Judgment, *Psychological Review,* 115, 2008, 155-185.

Appendix A: Documentation for Parser

---OUTLINE OF PARSER

Parser.m is a MATLAB function that takes as an input a file name. The program will assume that the file is a text file containing a knowledge base (KB). It will then access that file and do two things: check that the grammar for the KB is valid (using the rules found in the DSS Manual) and create data structures that house all of the data associated with the KB. Assuming there are no problems with the grammar, parser.m will create four .mat files and save it to the folder that the parser is in (One for each of the sections of the KB: parameters, functions, objects, and attributes). These files are only accessible via MATLAB either through function calls or the variable editor found in the GUI version. Attempting to access the data structures in any other way will only show gobbledygook. If during the parsing an error is detected, the program will print an error message that gives the file's name and line number before terminating early. In that case, no data structures will saved.

--- HOW TO RUN

To properly use parser, simply type into the MATLAB command line, "parser 'INPUT'.txt"- where INPUT is the name of the file. Ignore the double quotes, but keep the single quotes if there are spaces between words before the .txt. If there are no problems, MATLAB will flash the message "KB successfully parsed and loaded!". In order to view the newly created data structures, you need to use a separate MATLAB function called show. There are several possible ways to use show:

1. Type in "show 'all'" or "show 'KB'" this will list the names of all of the parameters, functions, objects, and attributes.

2. Type in "show 'ENTRY'" where ENTRY is the name of something you saw in the list generated by typing "show all". In this case, the function will list all of the entry's properties. If the object or attribute has synonyms, they can be entered instead.

3. If you want to see nested properties of a value of some attribute, type in "show 'values (ATTRIBUTE_NAME = VALUE)'". Where ATTRIBUTE_NAME is the name of the attribute and VALUE is the name of the attribute's value. If show is unable to find a match in any of these cases, it will display the message "Target Not Found". When entering names into show, remember that the function is case and character sensitive. Extra spaces will confuse the program

and lead to not making a match. Also, please remember to surround every thing past the word show with single quotes to make it one string.

---THE DATA STRUCTURES

Each of the four data structures is a cell array. I chose this approach (as opposed to a struct) due to the fact that some parts of the KB contain parenthesis. Such as: flexion(neck). MATLAB does not allow fields of structs to contain parenthesis. So, I used cell arrays instead. Although it is not as fast as looking up field names, we are able to preserve the information provided in the KB. Each entry in the cell array contains all of the relevant information such as name, synonyms, parts, and elements as structure fields. Here is an example of what one of the data structures look like:

ATTRIBUTES is a cell array. The 6th entry is blood pressure. Its fields are

Name: blood pressure

Isa: attribute

Setting Factor: False

Synonyms: Which is a cell array by itself. That cell array has only one cell, which is a structure. That structure has only one field called name whose value is "bp".

Value Type: Has an identical structure as synonyms. The only difference is that name is "ord".

Normal_Value: "Normal"

Values: This is a cell array with three cells. Each cell is a structure. And has the following names:

Name: Either decreased, normal, or increased.

Synonyms: (Only first and third cell have this) Synonyms is a cell array with one cell, which is a one-field struct. That one field either has the value hypertension or hypotension.

--- HOW TO RUN

To properly use parser, simply type into the MATLAB command line, "parser INPUT.txt"- where INPUT is the name of the file. If there are no problems, MATLAB will flash the message "KB successfully parsed and loaded!"

In order to view the newly created data structures, you need to use a separate MATLAB function called show. There are several possible ways to use show:

1. Type in "show all" or "show KB" this will list the names of all of the parameters, functions, objects, and attributes.

2. Type in "show ENTRY" where ENTRY is the name of something you saw in the list generated by typing "show all". In this case, the function will list all of the entry's properties. If the object or attribute has synonyms, they can be entered instead.

3. If you want to see nested properties of a value of some attribute, type in "show values (ATTRIBUTE_NAME = VALUE)". Where ATTRIBUTE_NAME is the name of the attribute and VALUE is the name of the attribute's value.

If show is unable to find a match in any of these cases, it will display the message "Target Not Found".

When entering names into show, remember that the function is case and character sensitive. Extra spaces will confuse the program and lead to not making the match.

---HOW THE PARSER WORKS

The parser does its work by iterating through each line in the KB. It will first check for a "mode" to be in (are we currently parsing parameters, functions, ect). Upon seeing the @ sign, it will know to switch modes. Once in a mode, it will call another function for a more detailed look at the one line. There is one function for each of the modes. (process_atrs, process_func, process_params, and process_obj)

In order to check the proper syntax for the grammar, I use a stack to store the delimiters. (This stack is really, a cell array, but I have another variable that tells which index we are at, and only use that index to view what is in the stack) Upon seeing a delimiter, I first check if there is a delimiter on the top already, and if it can be reconciled with the new one. Depending on the new delimiter, the program either: pushes it onto the stack, pops the stack, or does nothing further. Please look at the actual functions for specific examples.

There are some differences between how each of the data structures is created, but the basic idea remains the same. Each of the support functions will iterate through the line that was provided, and look at each character. If that character turns out to be a delimiter, then we execute the rules associated with it. It could be adding an item to a list, creating a nested property, adding a parameter to some entry. Before going any further, we collect the tokens (characters) between the current delimiter and the last one (or the start of the line). Those tokens are turned into a new character array, and sent to the appropriate location in the data structure.

How we figure out that appropriate location is one of the big differences between how each part of the KB is parsed. For process_objs, I used the stack previously mentioned. However, I needed a very different strategy for the attributes as its hierarchy is much more elaborate. What I did there is create a different structure called path. Path is a cell array where each entry corresponds to either a field name.

Appendix B: Transcript of Sample Parsing Sessions

(Note that EDU>> represents a new line in MATLAB's terminal. Each percentage sign represents a comment)

EDU>> %Parsing a KB that has an error leads to:

EDU>> parser KB.txt

Error in KB.txt at line 73

EDU>> %Having fixed the error at that line, we try again

EDU>> parser KB.txt

KB successfully parsed and loaded!

EDU>> %No problems, so now we can see the information through the show function

EDU>> show 'KB'

Displaying Parameters:

****

side

level

severity

Displaying Functions:

****

contralat

Displaying Objects:

****

patient

head

face

eyes

left eye

right eye

ears

neck

trunk

limbs

left upper limb

right upper limb

left lower limb

right lower limb

brain

cerebral hemispheres

left cerebral hemisphere

right cerebral hemisphere

right frontal lobe

left frontal lobe

right parietal lobe

left pareital lobe

right occipital lobe

left occipital lobe

right temporal lobe

left temporal lobe

corpus callosum

basal ganglia

right basal ganglia

left basal ganglia

diencephalon

right thalamus

left thalamus

brainstem

midbrain

pons

medulla

cerebellum

Displaying Attributes:

****

history

age

sex

examination(patient)

general examination

blood pressure

temperature

flexion(neck)

neurological examination

level of consciousness

respiration

eye exam

left optic disk appearance (left eye)

right optic disk appearance (right eye)

left retina appearance (left eye)

right retina appearance (right eye)

fundoscopy (eyes)

size (left pupil)

size (right pupil)

reactivity (left pupil)

reactivity (right pupil)

pupillary responses (eyes)

positions (eyes)

resting position(eyes)

spontaneous movements(eyes)

response to head turning

response to ice water calorics

left ice water calorics

right ice water calorics

motor exam

grimace(face)

posturing(limbs)

motor response to pain (limbs)

reflex exam

grasp reflexes

left grasp reflex

right grasp reflex

muscle stretch reflexes

left biceps reflex

left triceps reflex

left knee reflex

left ankle reflex

right biceps reflex

right triceps reflex

right knee reflex

right ankle reflex

plantar reflexes

left plantar reflex

right plantar reflex

EDU>> %Using show to view a parameter

EDU>> show 'side'

Displaying side

****

left

right

EDU>> %Using show to view a function...

EDU>> show 'contralat'

Displaying contralat

****

Name: contralat

Isa: Function

Argument Name: s

Argument Type: side

Rules:

left -> right

right -> left

EDU>> %Using show to view an object...

EDU>> show 'pons'

Displaying pons

****

Name: pons

Isa: Object

Instances: 1

Part Of: brainstem

Attributes: status

Elements: NA

Parts: left pons, right pons

Synonyms: metencephalon

Location(s):

((-40 40) (-100 -60) (-40 50))

EDU>> %Using show to view an attribute...

EDU>> show 'fundoscopy (eyes)'

Displaying fundoscopy (eyes)

****

Name: fundoscopy (eyes)

Isa: Attribute

Setting Factor: No

Value Type: mlt

Parts: left optic disk appearance, left retina appearance, right optic disk appearance, right retina appearance

Elements: NA

Values: papilledema, optic atrophy, hemorrhages

Normal Value: NA

Synonyms: funduscopic examination, ophthalmoscopy

EDU>> %Using show to view an attribute's value

EDU>> show 'values (fundoscopy (eyes) = papilledema)'

Displaying fundoscopy (eyes)

****

Displaying syn:

----

choked disks

swollen disks

Displaying elaboration:

----

pattern (sgl)

Displaying pattern (sgl)

****

Displaying values:

----

bilateral

left unilateral

right unilateral

Displaying bilateral

****

Displaying description:

----

left disk = papilledema

right disk = papilledema

Displaying left unilateral

****

Displaying description:

----

left disk = papilledema

Displaying right unilateral

****

Displaying description:

----

right disk = papilledema