An Introduction to Artificial Intelligence and the SAS® System
David B. Malkovsky, SAS Institute Inc., Cary, NC
H. John Maxwell, SAS Institute Inc., Cary, NC

ABSTRACT

This paper is a general survey of the field of Artificial Intelligence (AI) and concentrates on basic AI techniques, expert systems, and natural language processing. It begins with a definition of AI and compares it to conventional data processing. A few of the various application areas using AI techniques are discussed and methods of representing knowledge and reasoning using that knowledge are introduced. Some of the tools and problems associated with AI research and development are also discussed.

This paper emphasizes those areas of AI that are currently under investigation at SAS Institute, particularly the research effort that will culminate in SAS/EQL ™ software, a natural language query system that enables users to access data using only English.

WHAT IS ARTIFICIAL INTELLIGENCE?

There is wide disagreement in the field of artificial intelligence (AI) as to what constitutes the discipline. It is not unusual to find scientists who believe they are working in the field of AI who are not considered to be doing so by any of their colleagues. Conversely, there are scientists working in areas that are traditionally considered to be part of AI who refuse to apply that label to their work.

A helpful concept in understanding some of the confusion within the field is to classify AI researchers by a "means versus ends" test, where the research generally falls into one of two categories:

- Understanding the behavior of the human mind is the goal, and computers are simply computational tools for testing models of the mind.
- The design of more intelligent machines is the goal, and the behavior of the human mind is frequently a helpful model for simulating intelligence.

It would be easier to understand AI if a concise and generally accepted definition were available, but no such standard exists. A useful definition of AI is that it "is the part of computer science concerned with designing intelligent computer systems, that is, systems that exhibit the characteristics we associate with intelligence in human behavior - understanding language, learning, reasoning, solving problems, and so on." (Bart, Cohen, and Feigenbaum, 1981, Vol. 1, p.3)

CONVENTIONAL PROGRAMS VERSUS AI PROGRAMS

Describing the difference between a conventional computer program and an AI computer program is almost as hard as trying to find a definition for AI. To some people, any system written in LISP or Prolog is an AI program and everything else is conventional data processing. This is an oversimplification of the generally numeric nature of conventional programs and the generally symbolic nature of AI programs.

The algorithms and rules in a conventional program are embedded in the code and in the instructions that make up the program. A conventional program is usually involved with data storage, computation, and reporting. It is difficult, if not impossible, to understand the logic of a particular application by examining the code alone.

An AI program, however, usually has a definite separation between a set of rules (knowledge base) or general logical statements and a processor or engine that operates on those rules. This separation provides a means by which the "how and why" of the operation of an AI program can be presented to the user. AI programs also typically deal with implicit, incomplete, and noisy data whereas a conventional program operates with explicit and complete information.

AI APPLICATIONS

The field of AI, which is about thirty years old, covers a variety of research topics. The kinds of intelligent human behavior that AI investigators have tried to simulate include deduction, vision, learning, planning, problem solving, and natural language understanding. In addition, there has been research in basic AI techniques such as ways to represent knowledge. The development of special programming languages (for example, LISP, logic programming languages like Prolog, and object-oriented programming languages), tools (for example, expert systems shells) and architectures for AI has also been researched.

To date, the following kinds of AI applications have been developed:

- natural language interfaces to databases - interfaces that do not require users to learn special command languages
- expert systems - systems that use the knowledge of specialists to perform tasks such as medical diagnosis, chemical analysis, geological exploration, and computer system configuration
- automatic programming systems - systems to create programs based on descriptions of what the programs should accomplish
- vision and robots with sensory apparatus that respond to changes in their environment.

Such systems were developed for medicine, business, engineering, science, and manufacturing applications.

Basic AI techniques are usually divided into two main areas: knowledge representation and control strategy. One discovery made by AI research was that in order for a system to behave intelligently, it needs to have the ability to utilize a large amount of relevant knowledge. The goal of knowledge representation research is to devise representation schemes that can express all required information, can be stored and retrieved efficiently by machine, are understandable and easily updatable by human agents, and can be used to derive new or implicit facts. Control strategies are required to decide what information will be needed to solve the problems at a particular point or what alternative should be considered next. Because knowledge representation and control strategy issues are so pervasive in AI, it is worthwhile to consider them in a little more detail in the next section.
BASIC AI TECHNIQUES

Knowledge Representation

Knowledge representation (KR) has two aspects: content and formalism. As the names suggest, content has to do with what kind of information is stored and formalism has to do with the form in which the content is stored. For some applications, specialized knowledge of the problem is necessary. In others, common sense knowledge, such as everyday knowledge about how the world operates, is required. One area in which common sense knowledge is required is natural language understanding. Consider the following mini-story: Jane got a raise. She called the babysitter to see if she could come over that night and then got two tickets to Les Misérables. This story requires knowledge about the workplace, human emotions and motivations, ways of celebrating, and family life in order to be fully understood.

Each KR formalism has its strengths and weaknesses. It should be understood that KR is an area of active research and a general solution has not been found yet.

One of the earliest KR formalisms was the language of first-order logic developed by philosophers and mathematicians as a precise way of formulating proofs. (All men are mortal. Socrates is a man. Therefore, Socrates is a mortal.) It has the advantages of great expressive power and the property that anything derived from correct premises is guaranteed to be correct too. However, it has several disadvantages. One is that it does not model all kinds of human reasoning (such as probabilistic reasoning or reasoning from incomplete knowledge).

Other KR formalisms have been developed attempting to remedy these problems and to make first-order logic practical. One approach has been the development of logic programming languages such as Prolog. These languages may contain both a restricted version of first-order logic and some programming language control structures in order to be computationally efficient.

Another approach has been the development of KR formalisms that provide more structure than first-order logic (such as frame-based languages and rule-based or production systems). In general, the advantage of providing structure is that it makes it possible to perform certain kinds of automated deduction very efficiently. In addition, structure helps the programmer to modularize the problem. However, certain kinds of information are more difficult to express.

In a frame-based language, knowledge is organized as a collection of information about interconnected objects. Each kind of object can be represented in a structure called a “frame”. A frame has “slots” in which different kinds of information are stored. For example, a frame representing dogs might have slots for different attributes associated with dogs (such as hair color and name of owner). A frame can also have slots relating it to other frames; for example, a dog is a kind of animal. Then, attributes associated with the animal frame (for example, height and weight) can be “inherited” by the dog frame. Information about a specific dog like Fido would be represented by a dog frame in which the values of Fido’s attributes were filled in. Default values of attributes can be used to fill any slots for which the values were unknown.

In a typical rule-based system, knowledge is encoded as condition-action rules. The condition part describes a situation that must be true for the rule to be applicable, and the action part describes what action the system will perform if the rule is applied. The current situation is recorded in the production system’s “memory”. Every time its memory is changed, the system will check to see which rules are currently applicable. If more than one rule is applicable, some tie-breaking strategy is applied so that one rule is selected. After a rule has been selected, its action is performed. Types of action that it may perform include updating its memory and asking the user for information.

Control Strategies

Different control strategies are required for different knowledge representation formalisms and for solving different types of problems. Very often the domain and possible solutions for a problem can be thought of as a tree structure that must be searched through, and an important issue is how best to organize the search.

For example, one search strategy is called depth-first search. This strategy requires following a single branch of the tree until a solution is found or the branch fails. If no solution is found then the search continues by backing up to the nearest ancestor node that has not been expanded and continuing the search from that node.

An alternative approach is to examine all the offspring or children of the root node of the search tree. If one of them is the solution, stop the search. Otherwise, expand the new “leaves” and examine their successors. Repeat this until the required goal state is realized. This process is called breadth-first search. An obvious problem with this technique is the number of nodes expanded before reaching the goal state is likely to be prohibitively large.

Heuristics are devices that use additional information about the particular domain under study to make the control strategy more efficient and reduce the combinatorial explosions problems of the simple blind search strategies. For example, heuristic information can be used to decide which node of the tree to expand next and when certain nodes can be pruned from the search tree. In the first case, the search follows the path that seems the most promising or likely to result in the goal state. This is known as best-first search. In the case of pruning, heuristic information is used to determine when a particular node is not part of the answer and can be removed from further expansion.

As mentioned previously, rule-based systems are comprised of elementary pieces of knowledge (facts) and rules about how the various facts can be related. Inferences or new facts are made by applying rules to facts in order to produce new facts or hypotheses. Such a process usually continues until some predetermined goal such as the meaning of an English sentence or the next move in a game playing situation is reached. The two basic inferencing strategies for rule-based systems are known as forward and backward chaining. Forward chaining starts with all the available facts and deduces new conclusions or facts by applying any applicable rules until the desired goal state is reached. Forward chaining is also known as data-driven or goal-driven reasoning as it starts from an initial set of facts and proceeds forward to a set of consequences of the initial set of facts. Forward chaining is better for applications with an ill-defined goal or when there are a large number of possible goals. Planning and design problems, simulations, and configuration systems are typical uses of this technique.

A backward chaining inference or goal-directed reasoning process attempts to assert a goal as a new fact and tries to determine which other goals need to be proven to accept the initial hypothesis. If the supporting hypotheses are not already known facts, these hypotheses are added as new goals that must also be proven. This reasoning backward from the hypotheses to the data continues until all the goals can be substantiated. Backward chaining is most appropriate when there is not a lot of data and no clear place to start or when there is one particular question you want to answer. A medical diagnostic system is an example of using a goal-driven reasoning process.
All the reasoning strategies considered above depend on complete and perfect knowledge about the domain under study. That is, either a particular fact is known to be true or known to be false. Unfortunately, the real world is not so perfect. Probabilistic reasoning strategies are used when dealing with uncertainty, incomplete knowledge, or some amount of randomness. The rules of statistical probability theory and, in particular, Bayes' theorem are the tools that make it possible to represent likely but uncertain inferences.

Much of the reasoning in past AI systems has been monotonic in nature. That is, there exists some number of facts that are known to be true, and the number of known facts strictly increases over time as inferences or assertions are made. In domains that are changing or are based on incomplete information, it is sometimes necessary to both delete and add facts from a knowledge base. Systems that allow for such activity are called non-monotonic reasoning systems. These strategies (commonly known as truth maintenance systems) not only permit deleting a fact that was completely accurate at some previous time, but they also remove all other facts that are dependent on that fact. In such systems, care must be taken so that a large amount of processing time is not consumed with propagating changes.

EXPERT SYSTEMS

Expert systems (ES) belong to a broader category of programs known as "knowledge-based systems" and are usually rule-based. An expert system is designed to emulate the reasoning process of an expert in a particular domain or area of expertise. Expert systems are by far the best known and most successful commercial applications of AI to date.

Expert systems have been particularly successful in solving the following types of problems:

- diagnosis problems, in which the system determines the cause of a malfunction based on symptoms of the malfunction
- design and configuration, such as determining which components are required for a particular computer system
- planning and scheduling based on the constraints of time and materials
- simulation and process control
- general advice-giving problems, such as judging whether credit should be extended by a lender to a borrower.

These kinds of problems arise in many application domains, such as medicine, engineering, manufacturing, defense systems, and financial applications. Starting with the early research successes of the 1960s up to the present, more and more application domains have been recognized to which expert systems can be applied. Commercial application of expert systems is a phenomenon mainly of the last decade, and has been most successful in carefully restricted applications that involve diagnosis, configuration, or advice-giving. Numerous expert system building tools, usually called "shells", have appeared on the market for almost all hardware platforms, from PCs to mainframes. Although most of the original commercial applications were implemented on dedicated AI hardware, such as Lisp machines, there is now an overwhelming trend toward providing ES technology on standard platforms so that expert systems can be integrated into mainstream computing applications.

There are typically several different human participants in the creation of an expert system. The most important participant is the domain expert, whose knowledge and expertise is to be embedded into the program. Next in line is the knowledge engineer, whose job it is to encode the expert's knowledge and techniques in terms that can be used by the ES shell. If the shell is easy to use, the expert and the knowledge engineer can conceivably be the same person, although this is not the case for most of the shells currently available. Finally, there is an end user who will run the expert system after it is finished.

Although the architectures of expert systems can vary widely, most of them consist of the following components:

- A knowledge base consisting of facts and data. This knowledge base may be object-oriented, containing information about the relationships between data objects.
- If-Then rules for drawing inferences. This is where the bulk of the expert's knowledge will be encoded.
- An inference engine, supplied by the ES shell, that drives the rules.
- A user interface component for presenting the run-time version of the ES to the end user and for supporting the acquisition of knowledge during development.
- An explanation facility that can describe the reasoning the system uses to the end user.

Commercially available shells for building expert systems vary widely in capability and price. A number of shells have appeared for the PC that allow you to enter a limited number of If-Then rules and build simple applications. More expensive shells generally include extensive graphical user interfaces for both development and delivery systems. They may also feature some of the following:

- object-oriented or frame-based tools for dealing with hierarchies of concepts or objects
- both forward and backward chaining approaches to inference
- methods for performing inference in the presence of uncertainty
- Truth Maintenance Systems that allow for non-monotonic reasoning
- facilities to aid in knowledge acquisition, such as inductive methods that can derive rules from tables of example data
- connectivity features that allow for access to standard database management systems (DBMS)
- interfaces to conventional programming languages to allow for embedding expert systems in conventional applications.

Many of these features will be unnecessary for a given expert system application, but they are all useful in the right context. The last two items, which deal with interfacing ES to conventional applications, are receiving particular attention at the present time from both users and vendors. If ES technology is to be used extensively in the real world, the tools must be able to deal with real data and interface with conventional applications.

In the early 1980s there was considerable hyperbole in claims from the press and AI vendors concerning the power of expert system technology, which unfortunately led to some disillusionment in the industry when exaggerated promises could not be
many of the failures of ES have occurred because vendors and developers alike did not give enough attention to developing systems and tools that fit into conventional computing environments, but part of the problem has resulted from misconceptions about the nature of the intelligence that could be embedded in ES. The rules in ES capture the surface associations and relationships in a domain and not the underlying cause and effect. They consist of compiled knowledge (rules of thumb used by domain experts). As a result, ES fall when presented with novel situations not anticipated in the design of the system. Unlike humans, ES have no deeper understanding or knowledge on which to fall back. They cannot reason in some general fashion about what might be the cause of a particular problem. Nevertheless, ES have been very successful when applied to a domain that is sufficiently restricted so that the program can really capture the knowledge necessary for the task at hand. Many successful commercial ES have indeed been built in recent years, and there is great promise in this technology where it is applied appropriately.

**Natural Language**

The term "natural language" (NL) refers to languages like English, French, and Spanish that humans use to communicate with each other, as opposed to specialized "artificial languages" like FORTRAN and COBOL that are used to communicate with computers. Three most common applications of natural language are machine translation, text analysis, and information retrieval.

Machine, or automated, translation of texts from one natural language into another, was the first area to receive attention. The work started in the early 1950s and continues today. Unfortunately, it has not resulted in fully automatic and quality translation systems. Existing systems provide a first draft of the translation that is reviewed and modified by an editor.

Text analysis systems concentrate on the syntactic aspects of text. Such systems are used either as aids in writing (for example, finding grammatical errors) or in studies of authorship by analyzing the sentence structure or word usage of the text.

In the area of information retrieval the most successful natural language systems are front ends or interfaces for data base systems. At SAS Institute this is the area of AI research that receives the most attention and SAS/EQL software is the result of that activity. SAS/EQL software is a portable query tool that enables an individual using simple English to retrieve information from SAS data sets.

**Natural Language Query Interface**

The function of a natural language interface such as SAS/EQL software is to provide easy access to data without having to learn the syntax of a retrieval language and without having to know the structure and field names associated with individual data values. This ease of access does not come without an associated cost. Fortunately, this cost is paid only once by someone who is already familiar with the composition of the data. As is the case in expert systems where domain knowledge is entered by a domain expert, domain knowledge is required for the natural language system to function. This domain-specific information includes the content, concepts and context of the data that do not exist in the built-in knowledge base of SAS/EQL software or any similar product. The activity of acquiring domain specific information is known as "knowledge acquisition" or "database customization".

The primary task of knowledge acquisition is identifying what concepts are represented in a database and how these concepts relate to each other. This information includes how to join or merge data sets (using relationship paths between objects) and identification of hierarchies between objects for the purpose of generalizing or specializing types of attributes. For example, in a personnel database, to find a manager's salary requires generalizing the manager to an employee type where the salary field is most likely found.

Other aspects of database customization include the specification of domain-specific vocabulary, default printing formats, and the location of the data sets.

**Difficult Constructs**

Constructing a natural language interface is an interesting but frustrating undertaking, especially considering that a three-year-old child can do a better job of understanding English than computer programs that have required many years to build. A few examples will illustrate some of the inherent ambiguities of NL that make it difficult for a program to understand English.

- **List the average salesman's salary:** The typical response to this statement would be to calculate and list the average salary for all people with the job classification of salesman. Another plausible reading, however, would provide the salary figure for some entity identified as an "average salesman."
- **Who makes more than Bill does?** Comparatives can be difficult to understand because of missing information on the right-hand side of the comparison.
- **List the people who have children and live in Chapel Hill and Cary:** Conjunctions cause special problems because of their inconsistent use in English. In this sentence, the first and implies the presence of two characteristics (having children and living in a particular city) whereas the second and implies an or-ing of characteristics (living in either Chapel Hill or Cary but not in both locations). A similar problem exists for tie your shoes: The shoes are not tied up, but rather the shoe laces are being joined. This linguistic phenomenon is known as metonymy (a figure of speech where a part represents the whole or the whole represents a part).

**Stages of Processing**

In SAS/EQL software, there are six major stages of processing of a natural language query. They are:

- lexical and morphological analysis
- syntactic analysis
- semantic analysis
- pragmatic analysis
- human readable feedback
- translation.

The borders between these phases (syntax, semantics, and pragmatics) are not sharply defined and, in fact, there is some disagreement in the AI community concerning the definitions of each of these terms. For the purposes of discussing the processing of SAS/EQL software we will use our own definitions as they define the modular structure of the system.
The first stage of processing is the lexical and morphological phase. The lexical function of this phase categorizes each word into parts of speech such as noun, verb, adjective, preposition, or determiner by looking up the word in a lexicon or dictionary. A particular word can be more than one part of speech. For example, the word can is either a noun (garbage can) or a verb (can do). If a word is not found in the dictionary, the morphological processor transforms the word by looking for inflections of the word, such as the tense for verbs and plurals of nouns. Word prefixes such as non, which mark a word as having a negative feature, are also recognized. This phase also contains some general pattern matching and grammar. An ATN is similar to a finite state machine in which the transition between states is controlled by conditions on the arcs between the nodes. Specific words, word classes, or phrases will cause the transition from one node to the next. Actions associated with arcs store important linguistic information and build structures known as registers as the network is traversed. The registers can be thought of as variables in other programming languages and are used for guiding later processing of the sentence.

Nondeterminacy is a problem with parsing English sentences. In a simple ATN there is no facility to look ahead in the sentence to determine at a particular choice point in the network which of many syntactic paths is the correct one to pursue. That is, when looking at the conditions associated with a particular arc only the current word (or phrase) is used to determine the applicability of following a particular arc. For example, in the sentence The messy garbage can cover belongs to Oscar. If the current word can is its grammatical use as either part of a noun phrase or as a verb is equally plausible. The ATN used by SAS/EQL software is modified to provide a limited look-ahead feature, which reduces the amount of backtracking during parsing caused by such alternatives. With this look-ahead feature, the word belongs is recognized as the main verb in the sentence and thus can (and, for that matter, the word cover) must be part of the noun phrase.

The next stage of processing is semantics. Whereas syntax concerns itself with the relationship of words and phrases to each other, semantics is concerned with the meanings of words and phrases and the relationship between the objects the words represent. As is the case with words belonging to many grammatical categories, words can also have multiple meanings (polysemy). Consider these three sentences:

- Fred caught the football.
- Fred caught Jack's cold.
- Fred caught the fish.

In each instance, caught has a different meaning. The function of semantics is then to disambiguate various word senses. SAS/EQL software uses a case frame approach to relate the meanings of the syntactic constituents of the utterance. The association of meaning is done by matching various components of the sentence with various roles associated with a particular verb such as agent, beneficiary, and location. If the components for a particular frame are missing and they are deemed necessary, that frame or definition is rejected and another meaning or interpretation is sought.

Another function of semantics is the coercion of data types in some predetermined hierarchy. In a query such as Who manages the AI department?, the who must be coerced from a very general pronoun into an attribute that fits a slot in the manage case frame. First, who is coerced into an animate object type, next into a person type, and then it is specialized to a manager type, which is a specific type of person.

Ambiguity in a natural language query caused by modifier attachment is also handled by semantics. In a sentence like, Jack saw the scientist with a telescope, either Jack or the scientist could have the telescope. Semantics will present all the plausible alternatives to the later processing stages in a predetermined priority order. One method used in SAS/EQL software for ordering the interpretations is locality of reference; interpretations are ordered by the proximity of the modifier to the object being modified. By this heuristic, the first choice in our example is that the scientist has the telescope and the second is that Jack, using a telescope, saw the scientist.

The fourth stage of the processing of a query is called pragmatics. The pragmatics processor takes the case frame that is output by semantics and matches the classes of objects in it to entities and relationships in the actual physical data set. Pragmatics builds a structure called a 'logical form' which represents the query in terms of those entities and relationships. The functions performed in this phase include the mapping of time, space, and location to specific entities and deciding on the scope of quantifiers and operators.

In a query like Did Bonnie sell to a company in every state?, pragmatics must decide the scope of the quantifier every. Depending on the scope of the word every, two very different queries can result. The first reading involves looking for one particular company that has an office in every state that Bonnie sold something to, and the second is that Bonnie had a sale transaction in every state to any company located in that state.

Feedback to the user of the interface is the next stage of processing. As noted repeatedly, English is an ambiguous language and there is often more than one interpretation for a particular utterance. This stage of processing presents to the user a rephrasing of the query as it is understood by the program. Pure English is unsatisfactory because of its ambiguities and the actual database query language is of little use as this process is designed to insulate the user from such detail. Instead a form that is similar to English but that has explicit groupings and explicit operators is used. If the result is not what the user intended, the other plausible interpretations, if any exist, can be presented.

The last stage of processing translates the logical form into a programming language for the computer. SAS/EQL software uses SQL (Structured Query Language) queries to retrieve data. In this phase reporting fields are added, units of measure are converted, and aggregation of types (for example, an address is made up of street, city, and state components) is accomplished.
CONCLUSIONS

In this paper we have surveyed the field of Artificial Intelligence (AI), concentrating on basic AI techniques, expert systems, and natural language processing. We have emphasized those areas of AI that are currently under investigation at SAS Institute, particularly the research effort that will culminate in SAS/EQL software, a natural language query system that enables users to access data using only English.

REFERENCES


SAS is a registered trademark and SAS/EQL is a trademark of SAS Institute Inc., Cary, NC, USA.