AN OVERVIEW OF ARTIFICIAL INTELLIGENCE

by

DONALD J. GEMAEHLICH

B. S., Kansas State University, 1983

A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Electrical Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1984

Approved by:

E E Hall

Major Professor

TABLE OF CONTENTS

THIS BOOK CONTAINS NUMEROUS PAGES WITH THE ORIGINAL PRINTING BEING SKEWED DIFFERENTLY FROM THE TOP OF THE PAGE TO THE BOTTOM.

THIS IS AS RECEIVED FROM THE CUSTOMER.

THIS BOOK CONTAINS NUMEROUS PAGES WITH DIAGRAMS THAT ARE CROOKED COMPARED TO THE REST OF THE INFORMATION ON THE PAGE.

THIS IS AS RECEIVED FROM CUSTOMER.

LIST OF FIGURES

I. INTRODUCTION

Almost since the genesis of the first digital computer, computer scientists have dreamed a seemingly impossible dream, to create a computer which could mimic human thought. They wanted a computer which could find a problem, solve that problem, and then explain its solution to the problem.

In the 1960's, the integrated circuit was developed, making possible the development of more powerful machines. However, even these pieces of miniturized circuitry or their descendant of the 70's, the micro-computer, are not the bridge to the next generation of computers, the fifth generation. The next step is into the realm of artificial intelligence.

Intelligence

How can a machine be intelligent? Can a computer be "taught" to reason, to learn, to think without a programmer? The answers to these questions have been searched for since the dawn of the modern computer age; yet, a basic problem must first be solved. What is intelligence?

In the early days of computers, a machine was considered to be intelligent if it could display intellectual

powers equal to a human. These "electronic brains" were compared to the human brains in their abilities. John Von Neumann formulated comparisons associating data with human knowledge, a program's operation with decision making, the ongoing record of the program's operation with the stream of consciousness and the procurement of data with learning.[1] However, the relatively new theories of artificial intelligence have proven that these comparisons are totally without basis.

One of the easiest tools to use to study intelligence is the playing of games, since the processes used to play games model those of real life.[2] Computers can analyze many situations more quickly than a human can, and most early, "intelligent," game-playing computers used this ability to their advantage.[3] Using sophisticated programs to play chess and checkers, these machines could choose moves by looking twenty, forty, or more moves ahead to see if a winning situation resulted; however, these machines did not prosper because they were very slow and because they were not truly "intelligent."

In recent years, the criteria of intelligence has changed. The criteria includes the ability to solve general problems, the ability to use perception skills, such as sight and speech, the ability to solve "expert" problems, and the ability of learning by trial-and-error.[4] These four items are the basic abilities which currently set man apart from machine, and in limited cases a single

machine has been able to perform one of these tasks by using complex programs.[5]

Additionally, in order to think like a human, data must be accessed as it is in a human brain which requires a memory system with a different structure, a knowledge base system.

Only when all of these basic components of intelligence can be combined together into a single library of computer programs in the machine's memory can the machine respond coherently to human questioning, and intelligence be achieved.[6]

II. PROBLEM SOLVING METHODS

When a problem is to be solved, the goal state is the solution to the problem. Most problems are so large or so complex that a single process cannot lead straight from the initial state to the goal state without creating intermediate states or subgoals. These intermediate states represent the breaking of the main problem into smaller problems, each with its own goal.[7] Whenever an action is performed to achieve a subgoal, new actions become available to advance to a new subgoal. At each subgoal, new courses of action can be used: one state can lead to many. The map of these possible actions is known as a search tree with the root (the initial state) at the top of the tree and the branches (the possible actions) flowing downward. At the end of each branch is a node (a possible outcome) with one or more of these possible outcomes being the desired goal state[8] (see Figure 1).

In order to generate the search tree of a problem, a set of rules is used which describe possible actions. Each of these rules consists of a "right side" containing conditions of the current state and a "left side" which shows the new state after the action is performed.[9] At each node, the current state conditions are compared to

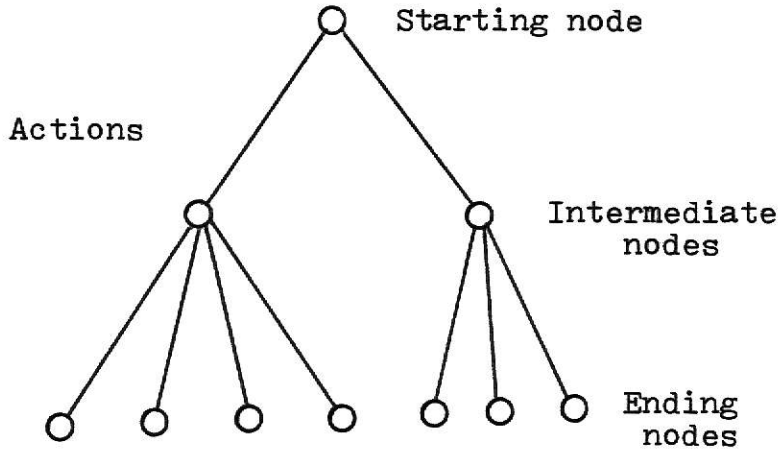


Figure 1. A sample search tree.

the "right side" of the rule, and for each "right side" that matches the current conditions of a node, a new node is generated using the "left side."[10]

In early programs, whenever a search tree was examined, the computer searched for the optimum answer. This search required looking over the entire search tree. This process was very slow in finding a solution and very wasteful of computer time, and in many computers the search tree was too large to feasibly search for the solution.[11]

Heuristic programming

Since finding the optimum solution is sometimes very difficult or in some cases impossible, most problem

solving routines use informal rules that do not guarantee that the best answer will be found, only that a very good answer will be found. These rules are known as heuristic principles.[12] Heuristic programming, basically, improves the efficiency of the searching of a search tree by choosing the most promising pathways early in the search and eliminating the less promising paths. The disadvantage is that the search process is incomplete.

Heuristic programs take the current state and all possible actions leading from the state, analyze the result of each action, and determine (usually by a scoring method) which path is most promising. The chosen path is then explored further. If it leads to a dead end, the program backs up to the previous node. If a new state is found, the process repeats. This type of programming may sometimes fail to notice an excellent path, but usually, good answers can be found without using the time needed to search all of the tree.

In addition to the time saving aspects of heuristics, other arguments can be found to support the use of heuristic programs in artificial intelligence systems. First, in most cases, an optimum solution is not required. Most people in most situations do not look for a best answer. Rather, if a solution that satisfies their requirements is found, the search is ended. Although heuristics are not very reliable in worst case situations, in the real world, worst cases are rarely encountered.[14]

Heuristic programming is an important tool of problem solving in artificially intelligent systems and is the corner stone of the following problem solving methods.

Search strategies

Heuristic search is a powerful technique that is used to solve many complex problems. However, a heuristic program needs a control scheme to determine how efficient the solving process actually is.

Generate-and-test. The simplest approach to problem solving is the generate-and-test strategy.[15] The steps to this strategy are 1) use the rules to generate a new node which could be a possible solution, 2) test the new node against the solution state, and 3) if the new node is a solution, stop the search. If it is not a solution, repeat the procedure. If this strategy is performed systematically, a solution will eventually be found. However, if the search tree is very large, the search process will take a very long time, or if a branch of the search tree has an infinite length, the process will never stop.[16]

The generate-and-test method has many forms which can simplify or complicate the search procedure. In its crudest form, without heuristics, the search tree is probed randomly, reducing the chance of finding a solution. At the other end of the spectrum, a totally controlled exhaustive, depth-first search will in most cases find a solution before random probing, but if the solution

state of the problem lies on the opposite side of the search tree from where the search started, the solution will not be found for a long time.[17]

Between these two extremes lies a search procedure, using heuristics, which can choose whichever path appears to be the most promising at that point. With the addition of backtracking capabilities, if a path starts to loop back upon itself or ends without a solution, the program can back through the search tree and find a new path to follow.

In simple problems, an exhaustive search procedure like the generate-and-test method can be useful; however, if the problem is more complex, the generate-and-test method becomes ineffective. Yet, if the search tree can be reduced in size by using other methods, such as limiting the depth of the initial search to a predetermined number of nodes (a depth-limiting search),[18] the generate-and-test procedure can be very effective in finding a solution. By combining search procedures in this way, the advantages of the separate techniques can be used advantagously while reducing each technique's disadvantages.[19]

Hill climbing. The hill climbing technique is a variation of the generate-and-test method. In this method, the heuristic function analyzes the current state and estimates how close a solution is and selects the action most likely to lead towards the solution.[20] Basically, the current state is tested to see if it is a solution, and if it is a solution, the problem is

solved. If it is not the solution, the rules of the problem are used to generate new nodes. These nodes are tested to see if any of them are solutions, and if one is, the problem is solved. However, if none of these nodes is a solution, the heuristics extimates which new node is more quickly moving towards a solution state. This node is made the next generating node and the procedure repeats.[21]

The hill climbing technique is not perfect. In some cases, the search procedure cannot find a move that satisfies the requirement to move towards a solution. First, the local maximum is a state which is closer to a solution than its neighbors, but it is not a solution. A second variety of problems is the plateau, or a group of neighboring nodes with approximately the same test value. A best direction cannot be determined from the data in each of these cases.

These problems can be solved by using variations of the hill climbing procedure. Backtracking to previous nodes can eliminate local maximum problems, and plateaus can be traversed by taking a set of jumps in the same direction, applying the same rule again and again. Yet, even with these methods, hill climbing is not practical for a large scale search. If another search method can localize a solution location, the hill climbing technique can more easily find the solution.[22]

Breadth-first. The two previous procedures, generate-and-test and hill climbing, are depth-first search procedures, the search progressed down a single branch until a solution was found or a dead end occurred. The breadth-first search technique, on the other hand, travels down each branch one node deep looking for a solution. If a solution is not found, the search progresses to the second layer of nodes, and these nodes are checked for solutions. This procedure continues until a solution is found.[23]

If the tree being searched has a solution that lies a finite distance from the start node, the breadth-first technique guarantees that the solution will be found. If a solution lies N nodes down in the search tree, the solution will be found when the path lengths from the start are N long. This technique also guarantees to find the solution that is closest to the starting node.[24]

The disadvantages of the breadth-first search procedure are numerous. First, since the number of nodes increases with every level explored, a lot of memory is required to store the data of each node. If the solution path is long, the work required also increases very quickly because each node at every level must be checked to see if it is a solution. Furthermore, if the search tree loops back upon itself, irrelevent nodes are stored, taking up memory space.[25] Finally, in search trees with long solution paths, a depth-first search method will probably find

the solution first.

Best-first search. The best-first search is a combination of depth-first search and breadth-first search.[26] The advantages of each of the two methods are kept and used to negate the disadvantages of the other.

In the best-first search procedure, the start node conditions are used to generate the first layer of successor nodes. A heuristic function is used on each successor node to determine which is the most promising. The most promising node is used to form a new set of nodes. These new nodes are analyzed and placed in a group with the other unused nodes, and the most promising of this set is used as the start node in the next expansion. This expansion process continues until a solution is found.

To illustrate, node A is the starting node of a search tree (see Figure 2). Using node A as the generating node, three nodes are generated and are analyzed. Of these three nodes, node B has the highest value, making it the most promising. New nodes are now generated from node B, creating nodes E and F; however, both of these nodes have a lower value than node D, making node D the new most promising node. The generation process is performed at node D, forming two more nodes. The values of G and H are placed with the values of the remaining unused nodes; C, E, and F. Of these nodes, F has the highest value; thus, the next generation of nodes would be from node F. This procedure continues until a solution is found.

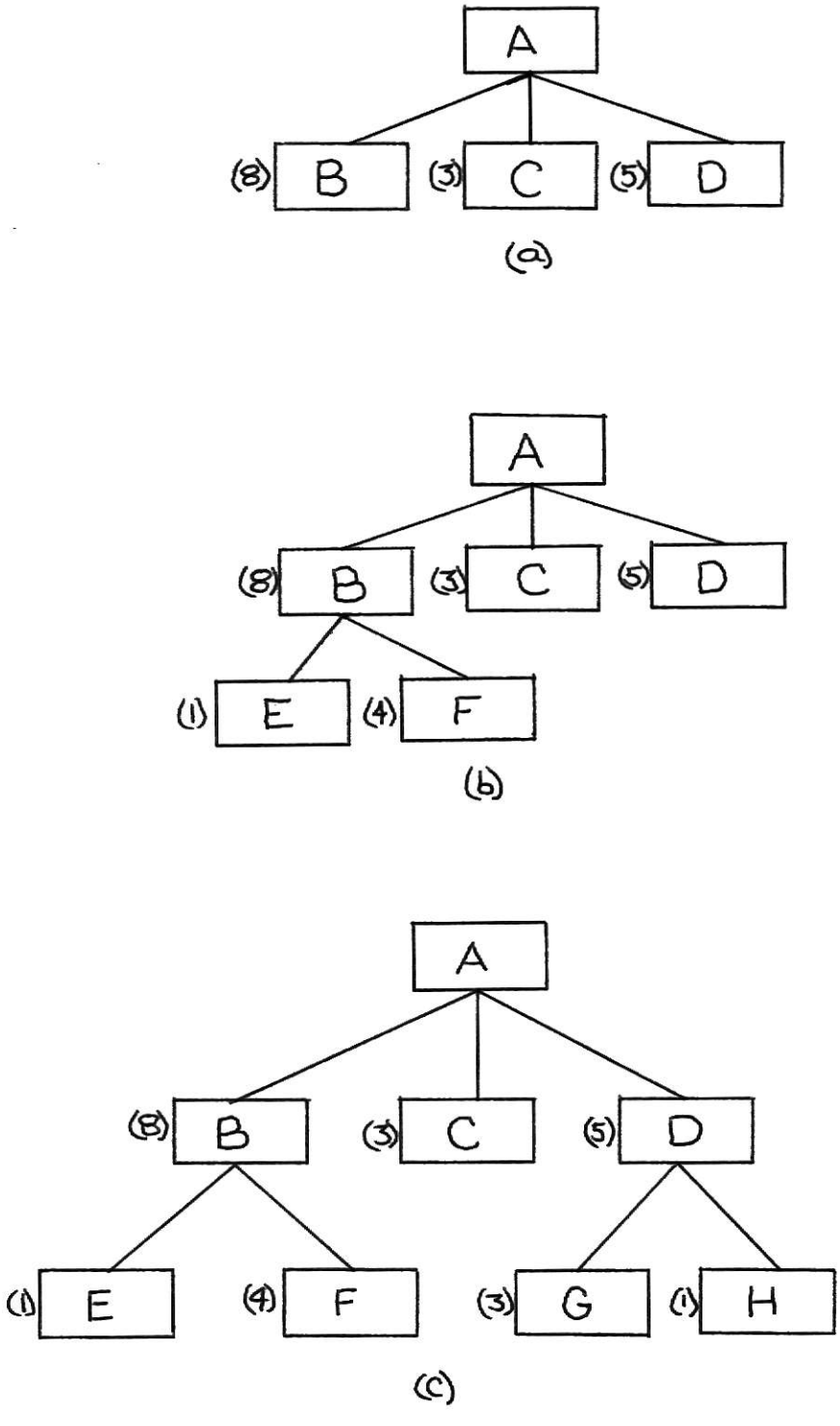


Figure 2. Best-first search process.

Minimax method. Game playing has been a fascination of artificial intelligence designers for years since the major strategies of games is to search for the best move. In one person games, such as a slide puzzle, a best-first strategy works the best. However, two person games become more complex; so, a more powerful technique is needed. This technique is the minimax search procedure.

In order to use the minimax procedure, a few assumptions are required about the problem. The following discussion deals with a conventional two person board game with the requirements: the players alternate moves, the moves are chosen from a known set, the loss of a piece by one player is considered a gain by the other player, and both players can "see" the board and examine moves.[27] These restrictions simplify the minimax method while still exposing its strong points.

The minimax search procedure is a depth-first, depth-limiting search procedure.[28] The starting node is used as the initial condition for a plausible-move generator (the set of moves for game pieces). These new nodes that are generated are analyzed using an evaluation test which assigns a numeric value to each node, and the best new position (usually the largest number) is noted. Since the evaluator is designed to return large numbers to indicate good positions, the basic result of this first step is to maximize the position rating.

In the next step, the program analyzes the opponent's

possible moves. Each node from the first step is used to generate new nodes (second level) representing the opponent's possible moves. These nodes are evaluated as before and their numeric values are recorded. However, in this case, the assumption is that the opponent will chose the best countermove for each of the first level moves, trying to minimize the first player's position. Thus, for each second level node group, the minimum value of each group is found and transfered backward to the corresponding first level node. Then, the new values of the first level nodes are to be maximized; so, the node with the maximum value is found, and the value is moved up to the starting node.[29]

For instance, Figure 3 shows a minimax process containing two layers of searching. New nodes are generated from the start node, and these nodes are evaluated. Since the maximum value is wanted, the node with a value of 9 is chosen. However, when the next layer of nodes are generated and evaluated, the new values are found. The second layer is the opponent's moves; so, the minimum values are chosen and moved back up the tree to the first level. The passed back values are then maximized, and the maximum value is chosen. In this case, the value is 1. This method shows that the best choice at one level may not be the best choice when looking farther ahead.

Minimax searching, however, is not foolproof. Its

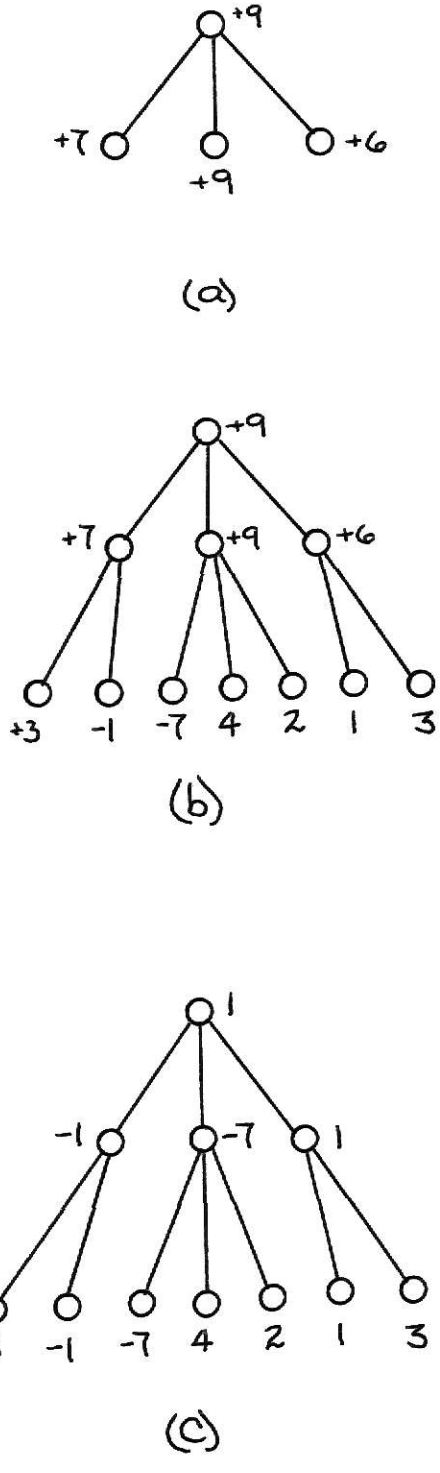


Figure 3. Minimax search process.

major drawback is the horizon effect, an occurrence in which inevitable bad moves creep out of view by various tactics. In chess, if many fronts are used to attack the computer's position, minimax programming can "lose" pieces that are in vulnerable positions.[30] Also, static evaluation techniques are used in this method; so if an unstable situation is present, such as a queen exchange, faulty values can be arrived at, and the wrong decision made.[31]

III. KNOWLEDGE REPRESENTATION

A major attribute of the modern computer is that large volumes of information can be accessed relatively quickly. The need for large volumes of information is a concern for artificially intelligent machines because as the machine learns, it needs more space to store the new knowledge. With this new knowledge, more problems can be solved, creating the need for yet more storage space. This upward spiral in needed storage is in conflict with the finite memory available to a machine.

To help solve this space problem and to make machine memories easier to access, the current database memory was changed into the new knowledge base memory. The knowledge base is made up of two main structures: an unstructured set of facts, and rules which can be used to determine new facts to solve problems.[32]

The architecture of the knowledge base system is also very different from the architecture of a database system. In a database system, the data storage is well structured with each piece of data having its own little pidgeon hole, and if the piece of data has an associated piece of data, the second piece has its own pidgeon hole. Both of these locations are looked after by the computer.[33] In a database system, the computer always knows which

pidgeon hole a piece of data is kept in. Conversely, the knowledge base is totally unstructured, and related items, such as data from an array, are considered as isolated pieces of data. The relationships between the isolated facts are determined as the problem is solved instead of being determined by the programmer.[34]

Monotonic representation

Inside a knowledge base, data can be represented in a number of ways: monotonic representation, nonmonotonic representation, and structured representation.

The first type of knowledge representation, monotonic, is a system of logic where known facts are used to create more facts. These facts are totally consistent with all previous facts; so the number of facts known to the machine is constantly growing. Every new fact which is learned or discovered will not change any old fact or rule.[35]

The basic form of monotonic structure is propositional logic. Propositional logic is relatively easy to deal with and decision making techniques have been developed to work with it. However, propositional logic has its drawbacks. For example, this type of representation is fine for single cases, but when a statement contains qualifications, the propositional system breaks down. The fact, "all men are mortal," is impossible to represent in propostional logic except by writing a seperate statement for each man.[36]

Since propositional logic does not function for all

cases, predicate logic is the next choice. With predicate logic, the basic facts are stored in predicate form. For example, the English sentence, "Socrates was a man.", would be represented as "Man(Socrates)". With this type of data representation only a small subset of all facts need to be entered into the machine. All other necessary facts can be determined by using the original facts and the rules of the knowledge base.[37]

The major problem with predicate logic is that the reasoning chain can be easily broken if even the smallest problem arises.[38] If logical statements are generated from English sentences, correct interpretations can be difficult to arrive at because English sentences have a tendency to be ambiguous. Furthermore, in some cases data can be represented in a number of ways with the best way of representation depending upon the facts used in the problem solving method. Even if the sentence can be represented clearly in predicate logic, the sentence set probably does not have enough information to solve the problem.[39] Predicate logic requires data that humans take for granted.

Nonmonotonic representation

The second variety of knowledge representation, nonmonotonic logic, differs from monotonic logic. Nonmonotonic representation is very useful in most real world problem domains which have incomplete information, changing situations, and a need of assumptions to solve

problems.[40]

The main difference between nonmonotonic and monotonic representations is that nonmonotonic logic has a changing knowledge base. If a piece of data contradicts a known piece of data, the knowledge base is changed, removing the old piece of contradictory data. Then the entire knowledge base must be checked to see which facts relied on the deleted fact. If these facts can be proved from other sources, they are kept. If the facts cannot be reproved, they are also deleted, and the process repeats.[41] This deletion process requires that each fact must have a listing containing the statements required in the creation of that fact.

As stated above, most real world problems do not give all of the required information. Nonmonotonic reasoning takes care of this problem by using default reasoning, a process of making sensible guesses when no contradictory evidence is present.[42] For instance, a person needs a gift for a friend. Since he knows that most people like chocolate, he assumes that his friend also likes chocolate since he has no evidence to the contrary. However, when the gift is given to the friend, the friend returns the gift and says that he does not like chocolate. This new data is added to the knowledge base, and all beliefs relying on the old assumption are discarded.

The computational definition of default reasoning

relates a lack of some information to a conclusion, but due to the nature of the knowledge base, all pieces of information are not stored explicitly.[43] This changes the relationship to "if X cannot be proved, then Y is true." However, proofs can become very long or seemingly impossible to finish without more precise information; thus the definition is again changed to state, "if X cannot be proved in some specified amount of time, then conclude Y."

The last definition appears to cause a large problem because if X cannot be proved in a set period of time, X is assumed to be unprovable. Then, whenever Y is used to prove another statement, the new statement could be considered invalid. Yet, default reasoning allows a program to assume a most probable case as long as no information is available to say differently.[44]

Additionally, nonmonotonic reasoning is useful since the real world is continuously changing. Information that was true an hour ago, may not be true an hour from now. A nonmonotonic system can change as information changes. If a once true piece of information is now found to be false, it and all of the information it created can be deleted, and the new piece of information can be added. A changing world can be more easily described by using a changing knowledge base.

A third advantage of using nonmonotonic reasoning is that in many real world problems, a beginning

assumption must be made in order to solve the problem. This assumption provides a starting point for the problem solving program, and if this assumption can be proved to be untrue, or if it does not provide a solution; the initial assumption is discarded, and a new assumption is made. With the new assumption, the routine starts the problem solving procedure over again.[45]

Nonmonotonic systems can be used very effectively in representing real world information, but this type of system requires more maintenance programming to remove inconsistent data and more memory to store the generated data.

Structured representation

Most facts can be stored in a knowledge base system, but some facts must be stored together because they are related. Information that needs to be stored together to be meaningful must be stored in a more structured form.[46] This type of structured storage is useful in representing knowledge of how to do sonething and in representing heuristic knowledge. The most common knowledge structures in knowledge bases are frames and scripts.

Frames are information storage systems that are used to describe objects. When a human comes upon a new experience, he does not build a new information structure on this experience alone. Instead, old memories of similar experiences are recalled with information from the new experience added wherever necessary. In a

knowledge base system, frames contain data on objects or experiences which can be built upon.[47]

Frames, usually, are used to describe stereotyped objects, such as a door or a table.[48] Each frame consists of a set of "slots" that can be filled with the subject's essential characteristics. Along with these characteristics can be a list of conditions that must be met for the object to fit the frame, or if data is missing, a default value is issued to fill the empty slot. For example, a table's default listing would include a rectangular, horizontal surface supported by four vertical members attached at each corner of the horizontal surface. When enough frames have been added to the knowledge base, the machine has the capability to infer concepts without being told.[49] For instance, since Fido is a dog, and since dogs like bones, Fido probably likes bones.

Frames can be used to reason by creating a temporary frame of the current situation with the situation's characteristics. The temporary frame's characteristics are compared to the other frames' characteristics to find a match. When a close match has been found, a new frame of the current situation can be easily made by recording all matching information and determining the different characteristics. With each new object or new situation that is encountered, a new frame needs to be developed about that object or situation.[50]

Scripts, on the other hand, are special-purpose

knowledge structures that can be used to summarize common human experiences.[51] From a low-level look, scripts and frames are identical since scripts, like frames, have slots that are filled by either data or a defaulted value. However, a script is different from a frame because the script deals with a stereotyped sequence of events instead of dealing with objects.

The important components of a script are 1) the entry position, conditions that must be satisfied before the remainder of the script can occur; 2) the result, the conditions that should be true after the script is finished; 3) the props, special objects required in the events of the script; 4) the roles, people who are involved in the script; 5) the track, variations on the general pattern that can occur; and 6) the scenes, the actual sequence of events. These components can be recorded in a script because in the real world, events follow set patterns.[52]

For example, a script on restaurants would describe the events of eating in a restaurant from entering, ordering, eating, and leaving. Using a script, a machine could infer that when John ordered a steak and paid for it at a restaurant, he most likely also ate the steak since that is the normal procedure. A script on restaurants can also help to resolve ambiguous statements which could lead to improper translation by the machine.

Like monotonic and nonmonotonic representations, structural knowledge representation is a useful part of

the knowledge base in artificial intelligence. Each has its own unique purpose, and the three together help to provide a solid system to represent the vast quantities of information in the real world.

IV. LEARNING

One ability that many artificial intelligence experts believe that a machine must have before it can be considered intelligent is the ability to learn. If a machine is unable to learn from experience, from its own mistakes, or from its own successes, then the machine is limited to using whatever knowledge the programmer gives it. Without the ability to learn, the machine will not be able to adapt to new situations or environments, and many people feel that a sign of intelligence in a being, or in a machine, is the ability to adapt to new situations.[53]

But why does an intelligent machine have to be able to learn? Without the ability to learn, a computer could not perform any action that was beyond its knowledge. For example, a machine can be given a program on how to unstack two stacked blocks, but what does the machine do when it is given the job of unstacking three stacked blocks? One reaction by the machine is that the task cannot be done because it has no knowledge of how to perform the task. This is the reaction of the dumb machine. If the machine has the capability to use its knowledge to reason, the machine can solve the problem of unstacking the three blocks, but without learning abilities, the machine has no way to save this knowledge itself. The machine can

find the procedure to unstack the three blocks, but each time this procedure is needed, the machine wastes time by solving the problem over again.

Just as with knowledge representations and problem solving methods, a variety to methods exist that allow a machine to learn.

Trial-and-error learning

Trial-and-error learning is just as the name implies. This method is learning by trial-and-error. If the generated procedure does not give the desired results, then the procedure is scrapped, and a new procedure is generated. This process is repeated until a workable procedure is achieved. When the procedure works as required, that procedure is saved for future reference.[54]

In a world composed only of blocks, three blocks are sitting on top of each other. Suppose the desired result is to have each of the blocks resting on the table. Further, the machine does not have a procedure to unstack three blocks, but it does know how to unstack two blocks. First, the machine develops a set of subgoals which could or could not reach a solution. The first subgoal developed by the machine is to unstack the middle block, but the middle block has a block resting on top of it; therefore, the two block unstacking procedure cannot work to achieve this subgoal. Since that subgoal cannot be used, a new subgoal is selected, unstack the top block. This subgoal can be accomplished since the top block rests on a block

and no block rests on top of it. This subgoal is recorded, and two blocks remain to be unstacked. However, the machine has a routine to unstack two blocks. When the goal state is reached, the machine stores the procedures that were used to unstack the blocks, and the machine now has the knowledge of how to unstack three blocks. Basically, the machine has randomly produced a set of procedures for the unstacking of blocks and has kept the one giving the desired result.[55]

Using the trial-and-error learning technique, the machine can also develop generalizations dealing with problem solving and can use the generalizations by asking if it (the machine) has seen this situation before.[56] This generalization can be used to solve one of many similar situations rather than having a special procedure for each situation. In the above case, if the machine was working with four blocks, a new procedure could be developed to unstack four blocks. On the other hand, the machine could recognize a similarity in subgoals between the three block routine and a four block routine. The machine sees that only the top block of the stack can be removed, creating a new procedure, "remove top block." A general procedure to unstack N blocks would be developed by executing "remove top block" N-1 times.[57]

Trial-and-error learning is a rather simple learning method, but it can be useful in developing new procedures to achieve goals.

Parameter adjustment learning

Where trial-and-error learning deals with developing new methods to achieve a goal state, parameter adjustment learning is concerned with altering the testing procedure of the heuristics. As described in the generate-and-test problem solving method, at each step of the problem solving process, the results are tested to see which is the best. This testing is accomplished by using a preprogrammed formula

$$\text{Sum} = c_1t_1 + c_2t_2 + \ldots + c_nt_n$$

where c_n is the weighting of the nth contributing feature and t_n is the nth contributing feature.[58] As the learning process progresses, the c values (the weighting of each factor) are changed until the optimum values are found.

In some cases, such as image recognition, parameter adjustment learning works very well since all of the weights of the terms which correctly predicted the results can be increased and all of the other weights can be reduced. Thus, the next time through the program, the image should be recognized more easily.[59]

However, not all programs that use testing functions are able to have their weights changed as easily as described above. This problem is very apparent in game playing programs since the entire game instead of single moves must be analyzed. Furthermore, what is adjusted if the machine loses, but always makes good moves, or if

the machine wins, but it makes bad moves? One method to solve this problem involves having the machine play itself with one side using the current weights and the other side using altered weights. Whichever side wins is assumed to have the better weight values.[60]

Parameter adjustment learning resembles the hill climbing problem solving method since the parameter values are progressing uphill in the direction of the most promising values. Because it resembles hill climbing, parameter adjustment also has the problems of the hill climbing method. Parameter adjustment learning is not a universal learning method, but it can be useful in many situations.

Concept learning

Concept learning, like the two previous learning methods, deals with learning in a specialized area, in this case, concepts. In concept learning, a concept is entered into the machine's knowledge base. This concept can then be used to classify other objects as being like that concept, or as not like that concept. The concept is started out in a very basic form. Then, by comparing the concept to different representations of the concept, a general concept structure can be developed.[61]

For example, the machine is shown a representation of an arch (see Figure 4). The program develops a basic concept of what an arch is. By analyzing the representation of the arch, basic characteristics can be found, such as an arch has two upright blocks sitting apart with a third

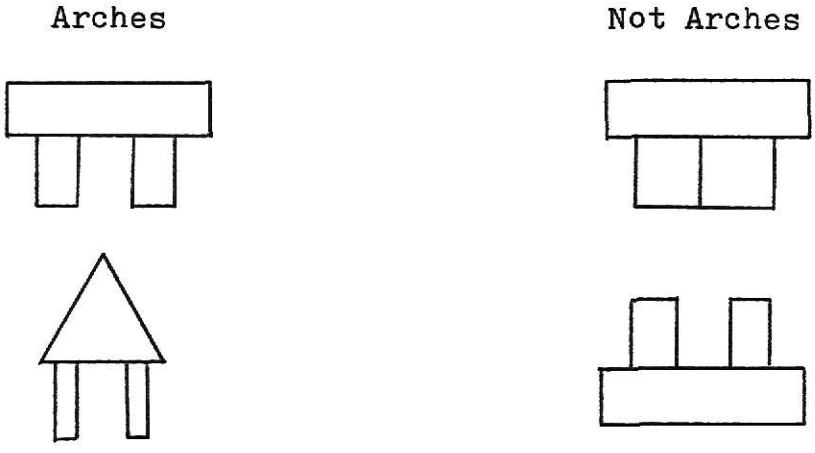


Figure 4. Concepts or arches.

block resting on top of the first two. By showing other representations of arches and representations that are not arches, a more concrete concept of an arch can be learned, and with this expanded concept, new variations of the arch should be easily discovered to be arches.

Concept learning is a useful tool in teaching machines about different physical atructures in the real world. If the teaching of the concept is concise enough, the machine will be able to recognize the structure, no matter what variations on the theme are present.[62]

V. NATURAL LANGUAGE UNDERSTANDING

The medium currently used to converse with computers is known as a computer language, but creating a program which allows a computer to understand a natural language (English, Japanese, French, etc.) is one of, if not the, most difficult charge of the artificial intelligence research today. The average toddler in the world knows very little about playing chess, but knows how to speak the native language: a feat computers are unable to master.[63] Some attempts at natural language understanding have been made in the past twenty years, but the simplest programs are limited to narrow fields of knowledge, such as in expert systems.[64]

The earliest work in natural language understanding dealt with translation machines. These machines were equipped with a blingual dictionary and with the syntax rules of the different languages, but these machines were a failure. In one case, the English sentence, "The spirit is weak, but the flesh is willing.", was translated into Russian and back into English with the result, "The vodka is strong, but the meat is rotten."[65] In another instance, researchers found references to a "water-goat" in a translation. Later, these researchers found that the

"water-goat" was the machine's translation for "hydraulic-ram."[66] Without the ability to understand the language, translation proved to be little more than gibberish.

In order to have a true natural language machine, the machine must be able to accept and give answers to requests in the user's natural language.[67] In addition, the user should be able to phrase the request any way he pleases. If part of the statement is ambiguous, the machine must find the ambiguity and ask the user to explain further. Currently, the user must know the quirks of the machine and its language, but the natural language ability would shift the burden from the user to the machine. With a natural language machine, the machine would learn the quirks of the user and his language.[68]

The major problem of a natural language system is the understanding of what the user is saying. First, the machine must understand a single sentence by understanding each word and then putting the words together to understand the entire sentence. The major steps are 1) syntactic analysis, transformation of a sentence into a structure showing how the words relate; 2) semantic analysis, assigning meanings to the structure; and 3) pragmatic analysis, the structure in reinterpretting to find out what is actually meant.[69]

Keyword matching

One of the simpest and earliest approaches to natural language understanding is keyword matching, combining the

three processes into one step. This procedure is accomplished by matching the input sentence with simple keywords stored in the memory. This approach also bypasses any true knowledge of the language; thus, unusual and ungrammatical languages can still be processed. The machine does not have an understanding of the conversation.[70]

An early program which uses keyword matching is ELIZA written by John Weizenbaum in 1966, which simulated a therapist.[71] This program relies almost entirely on a system of fixed responses. Each input response has a set of output responses which can be used; so, the program does not truly understand what is said. The input sentence is mapped directly to an output sentence and is then forgotten.

ELIZA has a script of primary keywords that provide the mapping ability to the output. For example, if the user mentions the word, "sister," ELIZA replies with one of its response sentences, such as "Tell me more about your family." However, if told "My friend's sister likes me.", ELIZA's response is again "Tell me more about your family." The inappropriate response is given because the keyword, "sister," appears and the word, "friend's," is ignored.[72]

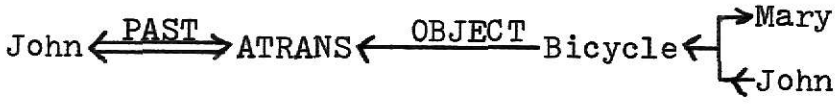
Even with this inefficient method of natural language understanding, keyword matching programs are very large, but not very practical in modern artificial intelligence machines.

Conceptual dependency

Conceptual dependency representation is a more complex system of natural language understanding than is keyword matching because conceptual dependency tears the sentence apart to find its meaning.[73] In conceptual dependency, the parsing, the separation of the sentence into its various parts, is set up around the main verb of the sentence. Since the representation of the verb is a low level representation, predication can also be analyzed to give greater understanding of the sentence.

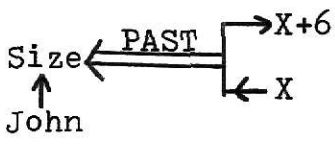
The basic procedure of conceptual dependency representation is similar to the parsing procedure taught in grade schools. The first step is the use of a syntactic analyzer to extract the main noun and verb from the sentence, and the analyzer also determines the type of verb. The three verb types are 1) MTRANS, actions of transferring mental information; 2) ATRANS, actions involving the transfer of possession; and 3) ATTEND, actions involving the senses.[74]

After these two steps are completed, the conceptual processor begins working by taking the verb and determining the correct usage of it. The conceptual processor takes the remainder of the sentence and attempts to parse it and place it in the empty slots of the verb structure. For example, if the sentence being analyzed is "John gave Mary a bicycle.", the structure would be represented as

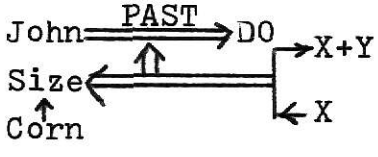


The use of conceptual dependencies also helps to show the differences between sentences. Sentences that on the surface appear to be similar, but have different meanings, have different diagrams (see Figure 5). Furthermore, the diagram of a sentence and the diagram of a paraphrased version of the sentence are similar even though the sentences may be very different.

This form of representation allows an easier form to represent natural language sentences, but, as with all natural language representations, a large knowledge base is required to hold all of the data needed.



"John grew six inches."



"John grew corn."

"John gave Mary a bicycle."

"Mary received a bicycle from John."

Figure 5. Conceptual dependencies.

VI. PERCEPTION

In order to make a computer more capable in the real world, the machine must be able to draw information directly from the environment by using the abilities of touch, sight, hearing, smell, and taste.[75] Of the five senses, sight and hearing have been the most studied and applied in artificial intelligence research. If robots and computers had better perception abilities, they would be more welcome in the work place.[76]

Perception research has been limited mainly to pattern recognition and speech understanding, but the major problems with both vision and hearing are that the amount of data needed to analyze a simple input signal is very large and ambiguities are very common.[77]

Speech recognition

With speech recognition, a number of problems exist. The perception of speech (a set of sound waves of varying frequencies) is the easy part. The hard part of speech recognition is grouping the sounds together to get meaningful sentences. For example, the sentence, "You gave the cat your dinner.", can be misunderstood as "You gave the catcher dinner." In current machines, the chance of a computer understanding a three word phrase is only

around 50%.[78]

Secondly, with speech recognition, absolute pattern matching cannot be used because two people do not say the same words in the same way. In fact, one person does not always say the same word in the same way. If a pattern recognition of a word's spectral frequencies is used, only relative matching could be used which reduces the accuracy of the speech recognition procedure.

The third major problem is that at most times more than one signal is present. Some machines have been able to understand single spoken words, but when the words are run together, as in normal speech, the accuracy drops off since there are no clear boundaries between words.

However, even with these problems, advancements in speech recognition have continued to occur. In speech recognition, the recognition procedure is divided into five steps: 1) digitization, 2) smoothing, 3) segmentation, 4) labeling, and 5) analysis.[79]

In digitization, the machine takes the input analog signal and digitizes it, creating a binary representation. The speech signal is sampled at a very high frequency (usually 20,000 Hz), and these samples are converted into a number with the higher amplitudes of the signal being represented as larger numbers.

The digitized signal is then inspected and smoothed. If a value of a sample is much larger or much smaller than

the values around it, the samples are altered. Since the real world deals mostly with continuous signals, the upward or downward spikes can be eliminated because they are probably caused by random noise.

The third step, segmentation, takes the smoothed values and combines similar values together. These new groups of values represent phones, individual sounds. The basic phonetic building blocks are assembled in this step.

The phones next are labeled. Basically, this step assigns a label to each phone with each label representing a different sound.

Finally, the analysis of the speech begins. The labeled segments are placed together in order to form a coherent statement. In this stage, the need for specific information about the domain is often required to determine accurate interpretations. However, in speech recognition, special effects, such as intonation patterns, can help define sentences and sentence segments.[80] Additionally, the analysis process can be made easier by finding "islands" (easily recognized words or phrases) from which to start the analysis. The "island" approach can prove useful since some words are normally pronounced more clearly than others.

Visual perception

Visual perception has the same problems as speech recognition. To identify a scene, the lines must be identified, and these lines must be grouped together to

form shpaes, objects, and shadows. Since a camera looks at a three-dimensional image and gives the machine a two-dimensional picture, ambiguous shapes can exist in pictures, such as lines being partially obstructed by other objects.[81] In other cases, pictures of the same object are shown at different distances, meaning that absolute matching of pictures will not work properly.[82] In this type of recognition, relative matching must again be used. Also, many objects must be analyzed at the same time since even in the simplest line drawing, parts of many objects can be obscured by other objects.[83]

In visual perception, the problem of identifying the various objects in a picture can be very difficult due to hidden lines, and possible perspectives of the objects. The amount of input data is very large and, unlike speech recognition, the possible variations of objects can be very large. However, by using a process known as constraint propogation, the number of possible combinations can be reduced by eliminating any cases that cannot occur.[84]

In constraint propogation, or constraint satisfaction, the first step is to identify each line. The machine has already found the lines in the picture and has its own drawing. Each line in the figure is identified as being 1) an obscuring edge, a boundary between objects or objects and background; 2) a concave edge, an edge between faces that points away from the viewer; or 3) a convex edge, an edge between faces that points toward the

viewer.[85] This identification process can be extended to include lines showing cracks and shadow lines.

Using the three line types described above (see Figure 6), objects can be described by labeling the object's vertices. Since three line types are present, a figure can be labeled in 3^N different ways, where N is the number of lines in the figure. For trihedral figures, there are four possible types of vertices (see Figure 7).

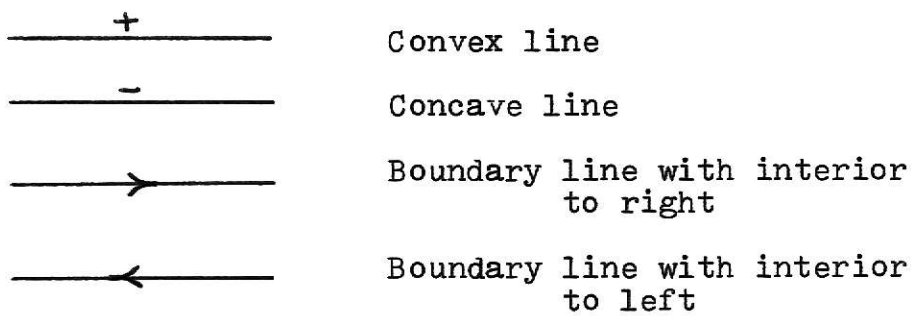


Figure 6. Constraint propogation lines.

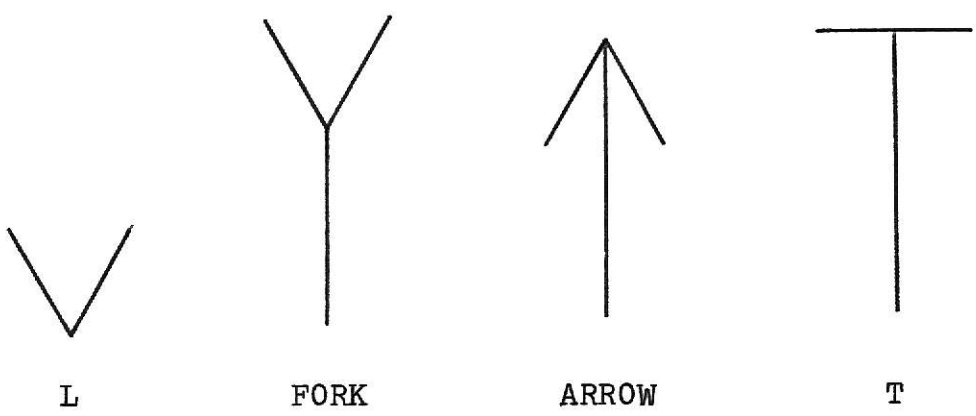


Figure 7. Constraint propogation vertices.

Examining the vertices, the number of combinations of lines can be determined. For the L vertex, two lines meet, giving 16 line combinations. Each of the three other cases; the arrow, the fork, and the T; have a three line connection; thus, they can be formed in 64 different ways. Addition of these numbers shows 208 different combinations for any given trihedral figure; however, only 18 are physically possible as shown in Figure 8.[86]

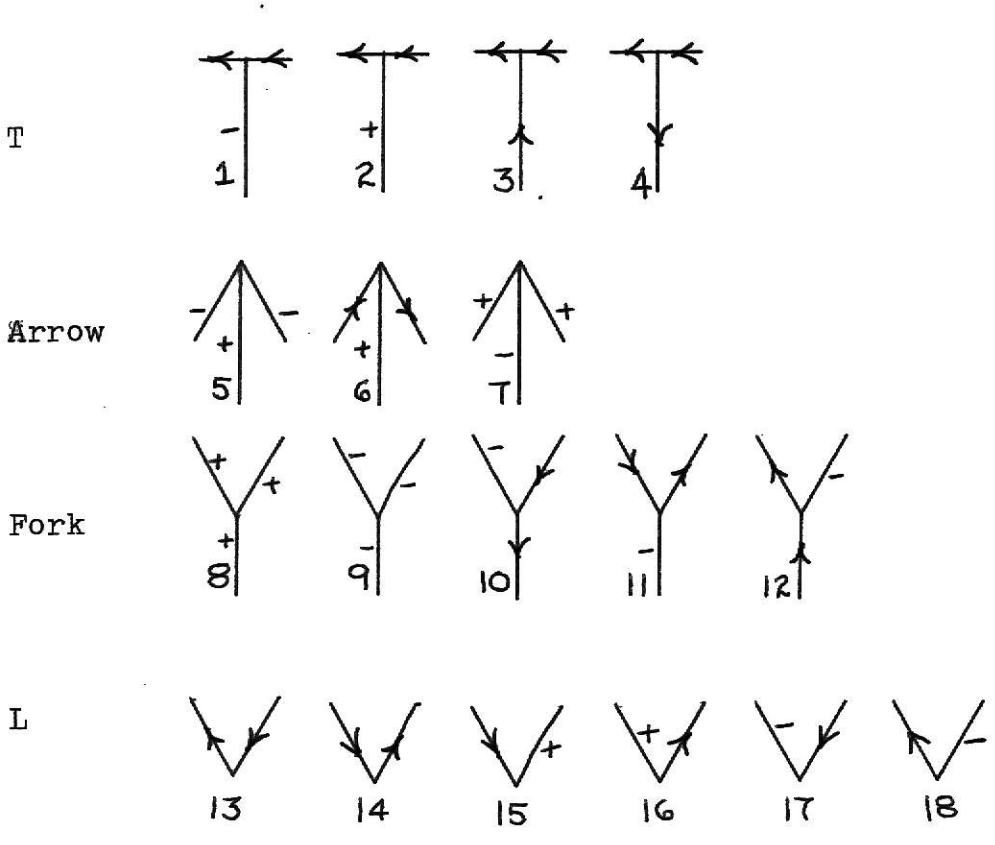


Figure 8. Possible vertices of constraint propogation.

For example, constraint propogation is used on Figure 9 to see if the object is consistent. First, the vertices are arbitrarily labeled with numbers, and all object-background baundaries are defined (see Figure 9b). Starting with a vertex, an attempt to find the vertex type is made. Since the interior of the object lies inside the angle, vertex 2 has only one angle type that fits the characteristic: Type 13. All vertices that can be labeled in this way are, labeling undefined lines with the appropriate labels (Figure 9c). The procedure is repeated on the remaining undefined vertices, and the newly labeled vertices can define more unlabeled lines (Figure 9d).

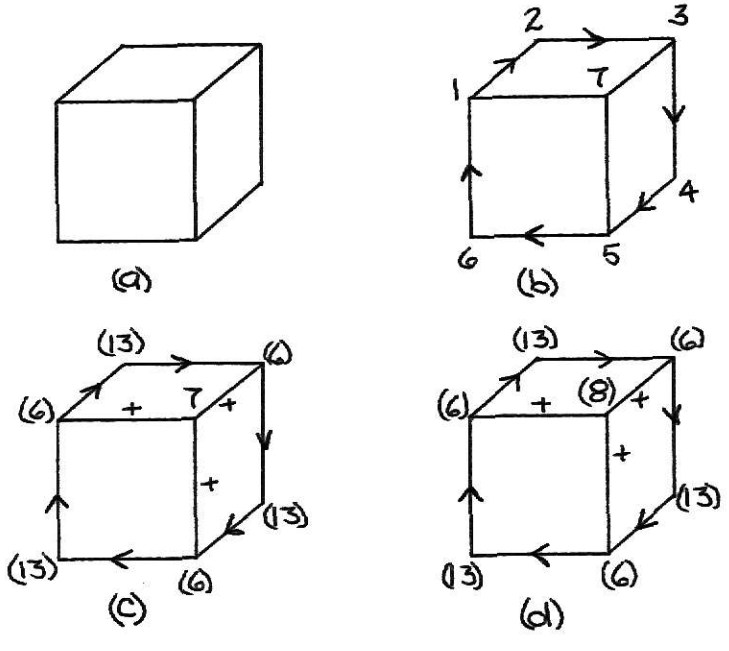


Figure 9. Object analysis using constraint propogation.

If an ambiguous figure is analyzed, the constraint propogation procedure will try to place two or more labels on a single vertex. Since this cannot happen in the physical world, the procedure is aborted.[87]

As with speech perception, the major problem lies in the large amount of memory required to analyze a scene, even in a simple block world. When the machine is given the ability to analyze the complex structures of the real world, the amount or memory required will also increase by as much as a few orders of magnitude.[88] However, these problems could be offset by giving the machine the ability to see better than a human. By giving the machine a camera able to see infra-red and ultra-violet, to see smaller things and farther away objects; the intelligent machine will be able to fit into its environment.[89]

VII. EXPERT SYSTEMS

The problem solving methods of the previous sections have been designed with an emphasis on "common sense" applications that almost any person could perform; however, there are a group of tasks that require a large amount of specialized knowledge which most people do not have. These tasks require the use of an expert system.

An expert system is a computer system that can perform a specialized task or a group of specialized tasks which only an "expert," who has acquired specific knowledge, could perform.[90] The expert system works in a similar manner to other artificial intelligence problem solving systems, but the knowledge base of the expert system contains very detailed information on a very small range of related subjects, the system's area of expertise. For instance, expert systems have been developed for use in the fields of medical diagnosis, electronic design, oil well problems, and even computer repair.[91] With the help of an expert system, human experts can make better choices.

Dedicated systems that fit the description of expert systems have been around for many years; however, expert systems have one major characteristic: they can explain their reasoning process.[92] This explanation is an important step since in many of the fields where expert systems are

employed, such as in medical diagnosis, the results probably would not be accepted without some evidence. The user of the expert system can be told the reasons why a certain procedure is proposed or not proposed.

The structure

The basis of the expert system lies in its knowledge base. The need for knowledge of a specialized area is so great that a very large knowledge base is created for a single expert system. This large amount of knowledge is a necessity because if not enough knowledge is available, the system cannot find accurate answers.[93]

Since the expert system relies extensively on its knowledge base, the structure of the knowledge is important. In early machines, the knowledge base was entirely unstructured, but modern research shows that a need for more structure in the knowledge exists. Additionally, the knowledge of the expert system falls into two catagories: the data of the knowledge base, and the program. The data of the knowledge base needs to be in a flexible memory structure to allow change at any time by the user. On the other hand, the program needs to be constant, being changed only in extreme cases. These two sections can then be controlled by simple procedures which vary with the situation.[94]

Reasoning processes

In expert systems, the reaching of a correct answer

is very important to the users of the system. This first step in the process of determining correct answers is the elimination of binary decision making, decisions with a yes or no answer.[95] In binary decision making, an item is either true or false. If the wrong answer is chosen, the result is completely wrong. Instead of determining what the absolute reason absolutely is, the expert system program deals with relative chances and likelihoods of a cause giving an effect.[96]

In the set of rules, which help to guide the search procedure, each rule is given a rating from 0 to 1. These fractional numbers represent the certainty of the rule being true, if the given data is true. For example, a rule states that if the car is John's, then (.99) the car is red. This rule means that the expert system can predict with .99 certainty (1.0 being perfect certainty) that any car owned by John is red. Rules, like the one above, can be combined, and by using algorithms, the certainty rating can be calculated for two or more rules used together.[97] By combining rules and certainty ratings, an expert system is able to take bits of inconclusive information and put then together to form a conclusive result. Additionally, the system can, by searching backward, find tests that should be conducted in order to help prove or to help eliminate possible answers to the problem.

The use of certainty ratings can also help in the determination of more than one answer. In some fields,

such as medicine, more than one answer is possible for a given set of data. With the help of certainty ratings, the two or three most likely answers can be determined, and the expert system can prescribe procedures to work with each result.

Expert systems are the basis for most artificial intelligence work because these systems can simulate the reasoning process of an expert dealing in a specific area of knowledge.[97] Furthermore, since expert systems are easier to duplicate (after the original is built) than humans can be made into experts, expert systems are beginning to be used in jobs where the number of human experts is very limited.

VIII. IMPLEMENTATION

In order to implement any of the artificial intelligence techniques described above, a special artificial intelligent machine should be constructed. This machine would have its own special language, one which can represent symbols instead of numbers. This new machine would also have to be built with many processors working in parallel to be able to think fast enough to operate the new programs.

Artificial intelligence languages

Basically, any program can be written in any language, but artificial intelligence systems require a language that can be used to work with both the data and the control procedures.

In early research, five system features of an artificial intelligence language were considered to be important: 1) the use of a large variety of data types, 2) the ability to decompose a process into small, independent parts for easier maintenance, 3) incorporation of flexible control structures, 4) the ability for interactive communication, and 5) the ability to produce efficient code.[98] These five major attributes have been incorporated in early artificial intelligence languages,

but in recent artificial intelligence research, additional characteristics have been found to be desirable: 1) the ability to work with lists, 2) the ability to find data relationships as the system runs, 3) the ability to perform automatic deductions, 4) the facilities to build complex data structures; frames, scripts, etc.; and 5) the use of goal-directed behavior control structures.[99] No existing language can provide all of these features, but some language can perform a few of the characteristics well at the expense of the others.

LISP is one of the earliest artificial intelligence languages and is the most important today. LISP is also the most widely used of the artificial intelligence languages.[100] The major reason for the widespread use of LISP in artificial intelligence research is that LISP is very efficient at handling symbols, particularly words and phrases.[101] Another important characteristic is that the symbols can be easily linked into data structures, such as scripts and frames, which are easier forms in which to store knowledge in a computer. Each data site also has pointers that show the direction to other lists, creating list processing.

However, the beauty of LISP is not entirely confined to its data structuring. LISP programs have the unusual capability to consider other LISP programs or even themselves as a set of symbolic concepts which can be altered.[102] Thus, LISP programs can be used to write other LISP

programs, or in relation to artificial intelligence, LISP programs can be used to alter themselves as they run. For example, a chess program using LISP would have the capability to rewrite programmed strategies depending on their effectiveness during the game.

Yet, LISP is not the only language that has been created for use in artificial intelligence machines. INTERLISP is a dialect of LISP that has the capabilities of LISP and that has a variety of data types, arrays and strings, in addition to lists.[103] INTERLISP also has the ability to run more than one routine at a time.

SAIL is the artificial intelligence language that is most similar to conventional high-level languages.[104] The language is reserved for areas in which conventional computing is required after an artificial intelligent solution is found.

PLANNER is a language built on top of LISP with its emphasis placed on goal-directed reasoning.[105] However, the language was never completely implemented because its reasoning process is limited to starting at the desired goal state and working backward to the initial state. This method can be very inefficient in some circumstances.

KRL is a language that is built upon INTERLISP. KRL has the ability to frame data structures very efficiently.[106] In KRL, each entity is represented as a unit with each unit having a set of information. However, in order to answer some questions, a KRL structure must store redundant

information. For example, in KRL the facts, "person 1 is the wife of person 2," and " person 2 is the husband of person 1," must both be stored. This type of redundant information can cause the required memory space to be prohibitively large.

Artificial intelligence hardware

Just as the languages of artificial intelligence machines must be specialized for the task, the artificial intelligence machine is also a very special device. LISP and the other artificial intelligence languages can be implemented on conventional mainframe; however, since the conventional mainframe is not designed to take advantage of the languages' special characteristics, the computing process is very inefficient.[107] In recent years, much research has gone into the design and construction of artificial intelligence machines.

For years, most computer designers have said that any procedure that can be processed in parallel processors can be processed in a serial processor. The serial processing will just be a bit slower. However, as the program becomes more complex, the serial processing becomes more than just a little slower. To increase computing capabilities of artificial intelligent machines, many processors can be placed in the machine to work at the same time.[108] One version of this scheme has four processors and nine memory units. If a processor needs data from memory, the processor requests the needed

data from memory. This type of processing speeds up the computing process because the processor controls the accessing of data rather than a controller.

The most radical design for artificial intelligence machines is being used by the Japanese on their fifth generation computer.[109] The design, known as dataflow, has a large number of identical processors working in parallel with each other. Unlike the conventional parallel machine which has a main processor controlling the program's operation, the dataflow machine has each processor working cooperatively with each other processor, eliminating the need for a central controller. However, the main problem with a dataflow machine is the division of the work between the processors. If the work is not divided evenly, one set of processors may be overworked, creating a data bottleneck, while other processors are idle, wasting computing time.[110]

In order to eliminate this problem, each piece of data is tagged. The tag tells the processor what operation the data is to be used in.

When a piece of data arrives at a processing unit, the data follows the path shown in Figure 10. First, the processing unit tries to match the new data's tag with the tage of another piece of data in the matching unit. If a match is found, the pieces of data are sent to the instruction-fetch unit. If no match is found, the data is stored for later use. The instruction fetch unit reads the

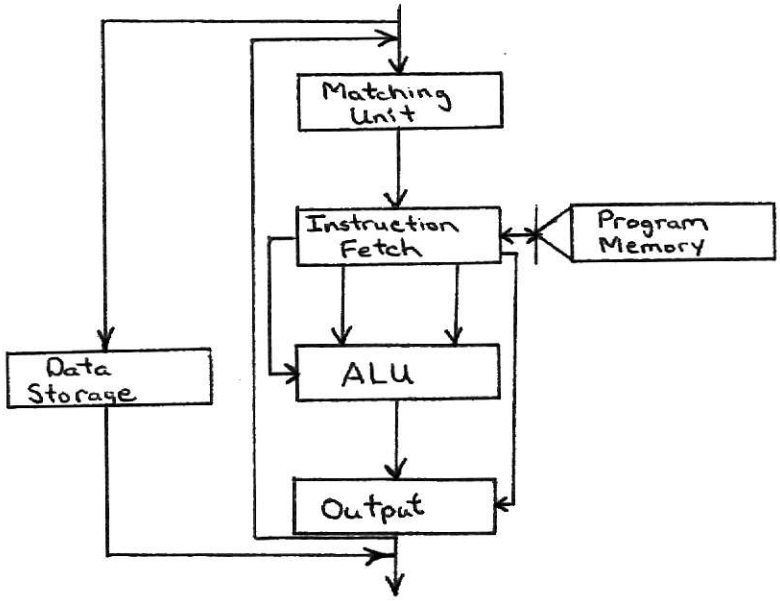


Figure 10. Dataflow structure.

tag on the data and determines what operation is to be executed on that data. The data is operated upon in the ALU, and the new data is sent to the output unit. In the output unit, the data receives the location of the next processing unit. Since each processor is working on whatever data is present, the dataflow processors are working a greater share of the time than the conventional machine's processors.[111]

The major problems with dataflow machines are in the transferring of data from one processing unit to another. If a piece of data must continually travel between opposite sides of the machine to be processed, the communication time lage can become very significant compared to the actual processing time. Additionally, as the number of processors increases, the cost of communications between each processor increases dramatically. For instance, to connect n processors, n^2 connections must be made.

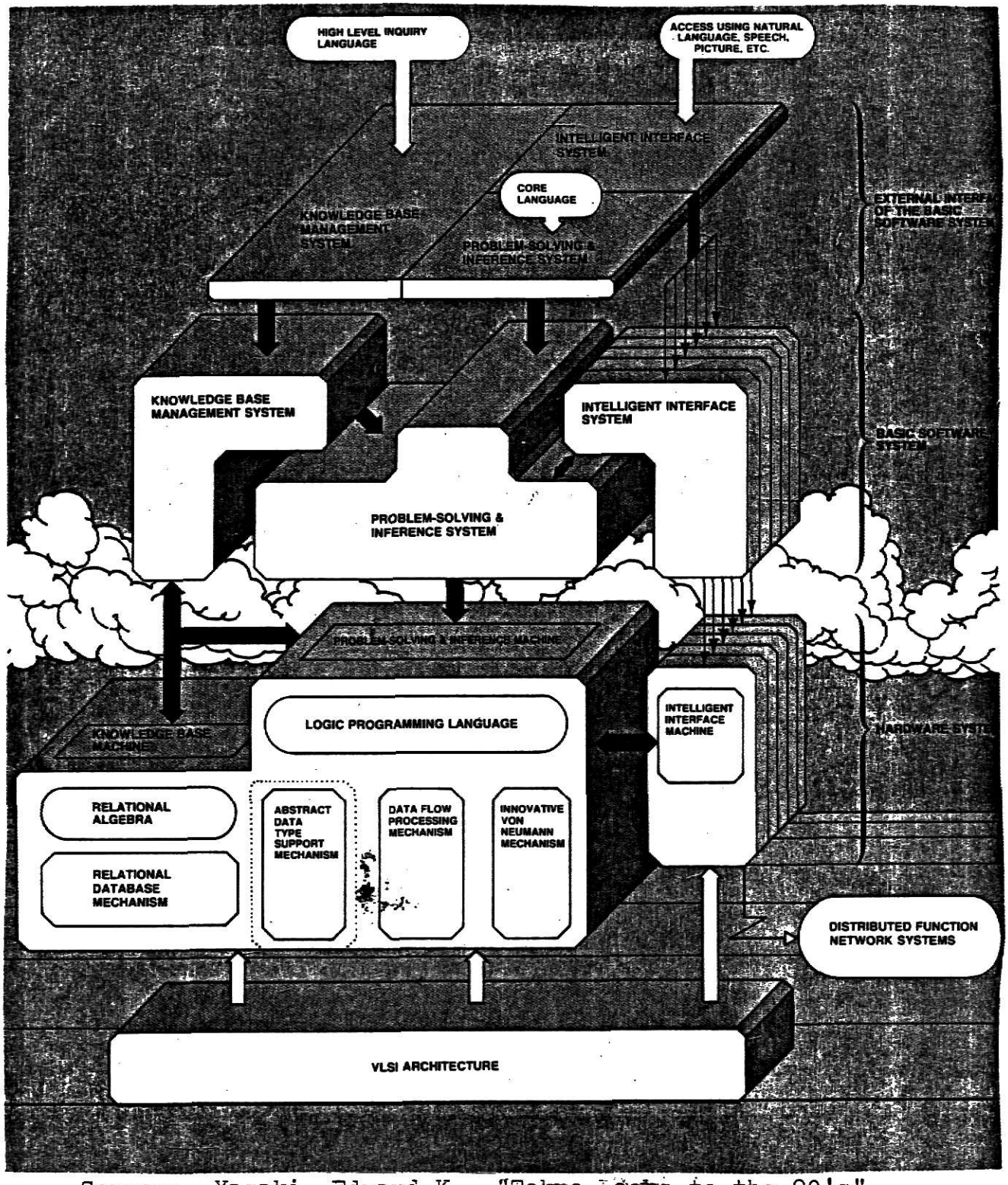
This growth of connections causes the cost to become prohibitive very quickly. Both of these problems can be solved by placing the processors in a ring structure and by allowing each processor to only "talk" with its immediate neighbors. With this arrangement, the number of connections increase linearly, and the number of long distance communications are kept to a minimum.[112]

Artificial intelligence machine

Up until this point, each charactersitic of the artificial intelligent machine has been discussed separately. Machines have been developed which can perform one or two of those characteristics, but for a machine to be intelligent, all of those characteristics must be integrated into its structure.[113]

The current concept of an artificially intelligent machine would incorporate all of those functions[114] (see Figure 10). This conceptual system would have an intelligent interface to the outside world with the capability to perform input-output operations with speech and graphic displays. It would be able to use natural languages and be able to translate these languages almost automatically. The intelligent machine would have access to a set of expert systems, and the various mathods to use them to solve complex problems. This system of expert systems would be built upon the basic problem solving machine with its knowledge base using both conventional and dataflow designs in the processing unit.

This machine will be the first intelligent machine to be created. This machine is the fifth generation computer.



Source: Yasaki, Edward K., "Tokyo Looks to the 90's", Datamation, Vol. 29, p. 112, July, 1983.

Figure 11. Concept of an artificially intelligent machine.

IX. CONCLUSION

The emphasis of artificial intelligence research has shifted in recent years. In the early years of artificial intelligence research, the goals were aimed at the long-term development of an intelligent machine, a pipe dream at that time. In the 1970's, the goal of most artificial intelligence researchers turned toward the design, the development, and the marketing of very profitable expert systems.[115] However, since the announcement of the development of the Japanese fifth generation machine, the emphasis of artificial intelligence research has again shifted to the development of an intelligent machine.

With this new emphasis on artificial intelligence, many questions have been asked. Is the building of intelligent machine morally right? Can man become a god by granting intelligence to whatever pleases him? Can an intelligent machine be "kept in its place?" However, the only question worth asking is "Can an intelligent machine be built?"

Experts cannot agree if an intelligent machine will appear, but whether that is today, ten years from now, or ten times ten years from tomorrow; artificial intelligence is one intellectual development that could benefit society by improving the quality of decision-making.

REFERENCES

FOOTNOTES

1. Waltz, David L., "Artificial Intelligence", Scientific American, Vol. 247, Oct 1982, p. 118.
2. Michie, Donald, On Machine Intelligence, John Wiley and Sons, New York, 1974, p. 21.
3. Rich, Elaine, Artificial Intelligence, McGraw Hill, New York, 1983, p. 2.
4. Rich, p. 4.
5. Michie, p. 67.
6. Michie, p. 52.
7. Hunt, Earl B., Artificial Intelligence, Academic Press, New York, 1975, p. 238.
8. Waltz, p. 118.
9. Rich, p. 31.
10. Ibid.
11. Rich, p. 34.
12. Waltz, p. 118.
13. Rich, p. 35.
14. Rich, p. 36.
15. Rich, p. 73.
16. Hunt, p. 230.
17. Rich, p. 74.
18. Hunt, p. 230.
19. Rich, p. 74.
20. Rich, p. 75.
21. Ibid.
22. Rich, p. 77.
23. Rich, p. 77.
24. Hunt, p. 266.

25. Rich, p. 77.

26. Rich, p. 78.

27. Hunt, p. 244.

28. Rich, p. 116.

29. Rich, p. 118.

30. Rich, p. 130.

31. Hunt, p. 252.

32. Harris, Larry R., "Fifth Generation Foundation", Datamation, Vol. 29, July 1983, p. 149.

33. Berry, Adrian, The Super-Intelligent Machine, Jonathan Cape Ltd., London, 1983, p. 127.

34. Harris, p. 149.

35. Rich, p. 176.

36. Rich, p. 138.

37. Harris, p. 150.

38. Rich, p. 140.

39. Ibid.

40. Rich, p. 176.

41. Ibid.

42. Ibid.

43. Rich, p. 177.

44. Rich, p. 177.

45. Rich, p. 178.

46. Alexander, Tom, "Teaching Computers the Art of Reason", Fortune, Vol. 105, May 17, 1982, p. 88.

47. Rich, p. 229.

48. Waltz, p. 132.

49. Alexander, p. 88

50. Rich, p. 223.

51. Alexander, p. 88.
52. Rich, p. 234.
53. Berry, p. 94.
54. Waltz, p. 123.
55. Michie, p. 15.
56. Michie, p. 67.
57. Waltz, p. 123.
58. Rich, p. 366.
59. Rich, p. 367.
60. Ibid.
61. Waltz, p. 123.
62. Waltz, p. 124.
63. Alexander, p. 86.
64. Waltz, p. 130.
65. Ibid.
66. Alexander, p. 86.
67. Harris, p. 154.
68. Ibid.
69. Rich, p. 304.
70. Rich, p. 305.
71. Waltz, p. 130.
72. Rich, p. 305.
73. Rich, p. 325.
74. Waltz, p. 130.
75. Hunt, p. 344.
76. Alexander, Tom, "Useless Science", Fortune, Vol. 105, May 31, 1982, p. 139.
77. Hunt, p. 345.

78. Berry, p. 113.

79. Rich, p. 349.

80. Hunt, p. 366.

81. Hunt, p. 346.

82. Hunt, p. 347.

83. Rich, p. 351.

84. Waltz, p. 126.

85. Waltz, p. 124.

86. Rich, p. 355.

87. Waltz, p. 128.

88. Hunt, p. 356.

89. Berry, p. 106.

90. Harris, p. 154.

91. "Artificial Intelligence, The Second Computer Age Begins", Business Week, March 8, 1982, p. 69.

92. Yasaki, Edward K., "A.I. Comes of Age", Datamation, Vol. 26, Oct 1980, p. 50.

93. Rich, p. 254.

94. Rich, p. 255.

95. "Artificial Intelligence, The Second Computer Age Begins", p. 69.

96. Staples, Betsy, "Computer Intelligence: Unlimited and Untapped", Creative Computing, Vol. 9, Aug 1983, p. 165.

97. Rich, p. 285.

98. Harris, p. 156.

99. Rich, p. 390.

100. Myer, Edith, "Machines that LISP", Datamation, Vol. 27, Sept 1981, p. 105.

101. Alexander, Tom, "Computers on the Road to Self-Improvement", Fortune, Vol. 105, June 14, 1982, p. 149.

102. Ibid.

103. Rich, p. 395.

104. Rich, p. 396.

105. Hunt, p. 281.

106. Rich, p. 397.

107. Rich, p. 405.

108. Marbach, William K. and Cook, William J., "The Race to Build a Supercomputer", Newsweek, Vol. 102, July 4, 1982, p. 60.

109. Yasaki, Edward K., "Tokyo Looks to the 90's", Datamation, Vol. 29, July, 1983, p. 113.

110. Lerner, Eric J., "Data-flow Architecture", IEEE Spectrum, Vol. 21, April 1984, p. 57.

111. Lerner, p. 59.

112. Lerner, p. 60.

113. Michie, p. 67.

114. Yasaki, "Tokyo Looks to the 90's", p. 113.

115. Bass, Alison, "Artificial Intelligence in a Rut", Technology Review, Vol. 86, Aug-Sept 1983, p. 82.

BIBLIOGRAPHY

Alexander, Tom, "Teaching Computers the Art of Reason", Fortune, Vol. 105, pp 82-92, May 17, 1982.

Alexander, Tom, "Useless Science", Fortune, Vol. 105, pp. 139-145, May 31, 1982.

Alexander, Tom, "Computers on the Road to Self-Improvment", Fortune, Vol. 105, pp. 148-160, June 14, 1982.

"Artificial Intelligence, The Second Computer Age Begins," Business Week, pp. 66-75, March 8, 1982.

Bass, Alison, "Artificial Intelligence in a Rut", Technology Review, Vol. 86, p. 82, Aug-Sept 1983.

Berry, Adrian, The Super-Intelligent Machine, Jonathan Cape Ltd., London, 1983.

Harris, Larry R., "Fifth Generation Foundations", Datamation, Vol. 29, pp. 148-156, July 1983.

Hunt, Earl B., Artificial Intelligence, Academic Press, New York, 1975.

Lerner, Eric J., "Data-flow Architecture", IEEE Spectrum, Vol. 211 pp. 57-62, April 1984.

Marbach, William D., and Cook, William J., "The Race to Build a Supercomputer", Newsweek, Vol. 102, pp. 58-64, July 4, 1983.

Michie, Donald, On Machine Intelligence, John Wiley and Sons, New York, 1974.

Myer, Edith, "Machines that LISP", Datamation, Vol. 27, pp. 164-166, Aug 1983.

Rich, Elaine, Artificial Intelligence, McGraw Hill, New York, 1983.

Staples, Betsy, "Computer Intelligence: Unlimited and Untapped", Creative Computing, Vol. 9, pp. 164-166, Aug 1983.

Waltz, David L., "Artificial Intelligence", Scientific American, Vol. 247, pp. 118-132, Oct 1982.

Yasaki, Edward K., "A.I. Comes of Age", Datamation, Vol. 26, pp. 48-54, Oct 1980.

Yasaki, Edward K., "Tokyo Looks to the 90's", Datamation, Vol. 29, pp. 110-115, July, 1983.

AN OVERVIEW OF ARTIFICIAL INTELLIGENCE

by

DONALD J. GEMAEHLICH

B. S., Kansas State University, 1983

AN ABSTRACT OF A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Electrical Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

ABSTRACT

The dream of artificially intelligent machines has been around for as long as computers have, but not until now has the technology been equal to the task. Artificial intelligence is the terminology used to describe the machine's ability to think like a human. With the ability to store information as the human brain does, to reason through problems, to communicate in the native language, to perceive data from the environment, and to learn from its mistakes; the artificially intelligent machine will be able to adapt to the changing world. Using designs for computer hardware and new languages to run that hardware, the time for intelligent computers is not too far away.