If a cat can kill a rat in a minute, how long would it be killing 60,000 rats? Ah, how long indeed! My private opinion is that the rats would kill the cat.

--- Lewis Carroll, on the advantages of parallelism

1. Introduction

1.1 Preview

The human mind can do many remarkable things. Of these, perhaps the most remarkable is the mind's ability to store a huge quantity and variety of knowledge about its world, and to locate and retrieve whatever it needs from this storehouse at the proper time. This retrieval is very quick, very flexible, and in most cases seems almost effortless. If we are ever to create an artificial intelligence with human-like abilities, we will have to endow it with a comparable knowledge-handling facility; current knowledge-base systems fall far short of this goal. This report describes an approach to the problem of representing and using real-world knowledge in a computer.

The system presented here consists of two more-or-less independent parts. First, there is the system's parallel network memory scheme. Knowledge is stored as a pattern of interconnections of very simple parallel processing elements: node units which can store a dozen or so distinct marker-bits, and link units which can propagate these markers from node to node, in parallel through the network. Using these marker-bit movements, the parallel network system can perform searches and many common deductions very quickly: the time required is essentially constant, regardless of the size of the knowledge-base. The network is similar to the parallel marker-propagating

network proposed by Quillian [1968, 1969], but is much more tightly controlled. This system is presented as a faster, more effective, and much simpler alternative to the currently popular approach of using domain-specific meta-knowledge, in the form of local procedures, to guide and limit serial searches in a large knowledge-base.

The second, more traditional part of the knowledge-base system is a vocabulary of conventions and processing algorithms -- in some sense, a language -- for representing various kinds of knowledge as nodes and links in the network. This set of conventions is called NETL. In many ways, NETL is similar to existing systems for representing real-world knowledge, such as the partitioned semantic networks of Hendrix [1975a, 1975b, 1976] and the frame-based KRL system [Borow & Winograd, 1976], but it differs from existing systems in three respects:

First, NETL incorporates a number of representational techniques -- new ideas and new combinations of old ideas -- which allow it to represent certain real-world concepts more precisely and more efficiently than earlier systems.

Second, NETL is built around a single, clear, explicit organizing concept: an effective knowledge-base system, in addition to storing and retrieving isolated facts, must provide the user with the ability to create and use *virtual copies* of descriptions stored in the memory. By "virtual copy", I mean that the knowledge-base system *behaves* as though a portion of the semantic network has been copied (with some specific alterations), but it does not actually create the redundant structure in memory. The descriptions that are copied in this way may be arbitrarily large and complex, with parts, sub-parts, and internal relationships. This entire structure is inherited by the copy, not just a few global properties. NETL is not unique in *providing* such a virtual copy facility, but it is unique in *stating* this goal explicitly, in clear and simple terms, and in relating all of its representational machinery to this goal. This adds considerably to the conceptual clarity and unity of the resulting system, and it provides us with a way of determining

whether the system's accessing mechanisms do what they are supposed to do.

Finally, NETL has been designed to operate efficiently on the parallel network machine described above, and to exploit this machine's special abilities. Most of the ideas in NETL are applicable to knowledge-base systems on serial machines as well.

A simulator for the parallel network system has been implemented in MACLISP, and an experimental version of NETL is running on this simulator. A number of test-case results and simulated timings will be presented.

1.2 The Knowledge-Base Problem

Suppose I tell you that a certain animal -- let's call him Clyde -- is an elephant. You accept this simple assertion and file it away with no apparent display of mental effort. And yet, as a result of this transaction, you suddenly appear to know a great deal about Clyde. You can tell me, with a fair degree of certainty, how many legs he has, what color he is, and whether he would be a good pet in a small third-floor apartment. You know not only that he has eyes, but what they are used for, and what it implies if they are closed. If I try to tell you that Clyde builds his nest in a tree or that he is a virtuoso on the piano or that he amuses himself by hiding in a teacup, you will immediately begin to doubt my credibility. And you can do this very quickly and easily, with none of the sort of apparent mental effort that would accompany, say, adding two four-digit numbers. This effortlessness may be an illusion, but it is a compelling one.

"Elephant", of course, is not the only concept that behaves in this way. The average person knows a huge number of concepts of comparable or greater complexity -- the number is probably in the millions. Consider for a moment the layers of structure and meaning that are attached to concepts like lawsuit, birthday party, fire, mother, walrus, cabbage, or king. These are words we use casually in our daily lives, and yet each of them represents a very substantial package of information. In technical fields (except, perhaps, for the more austere parts of mathematics) the situation is the same. Consider how much you would have to tell someone in order to fully convey the meaning of concepts like meson, local oscillator, hash-table, valence, ribosome, or leukemia. And yet, once these concepts are built up, they can be tossed around with abandon and can be used as the building blocks for concepts of even greater complexity.

The point is not just that we can handle large chunks of knowledge as though they were atoms; the important thing is that we can find our way through these complex, nested structures to whatever individual fact or relationship we might

need at any given time, that we can do this in a very flexible and efficient way, and that we can somehow avoid having to look individually at each of the vast number of facts that could be -- but are not -- relevant to the problem at hand. If I tell you that a house burned down, and that the fire started at a child's birthday party, you will think immediately of the candles on the cake and perhaps of the many paper decorations. You will not, in all probability, find yourself thinking about playing pin-the-tail-on-the-donkey or about the color of the cake's icing or about the fact that birthdays come once a year. These concepts are there when you need them, but they do not seem to slow down the search for a link between fires and birthday parties. If, hidden away somewhere, there is a sequential search for this connection, that search is remarkably quick and efficient, and it does not become noticeably slower as the knowledge base expands to its adult proportions.

This impressive ability to store and access a large and diverse body of knowledge is a central feature of human intelligence. The knowledge-base system provides essential support for the other components of intelligence: the peripheral processes that handle such things as vision and speech understanding, and the linear, sequential, conscious kinds of thinking that characterize our problem-solving behavior. The knowledge base is the common ground for these diverse elements, the glue that holds everything else together.

It follows, then, that any *artificial* intelligence, if it is to be even remotely human-like in its capabilities, must include a knowledge-base system with abilities comparable to those possessed by humans. To date, in the field of AI research, we have been unable to achieve or even approach this goal. We can make -- and have made -- a certain amount of progress toward understanding the sensory and problem-solving areas of thought by confining our investigations to relatively knowledge-free problem domains -- tasks like cryptarithmetic puzzles and the symbolic integration of mathematical expressions. We can make still more progress by patching together tiny knowledge bases,

just sufficient to serve as scaffolding for whatever test cases we are working on at the moment. But until we can find an adequate solution to the knowledge-base problem, all of our work will be fragmented and somewhat distorted. Sooner or later, we will have to confront that elephant.

The problem is not that we are unable to store and retrieve enough *explicit* knowledge -- that problem was solved long ago. In the property lists of LISP, in the hash-tables of LEAP and SAIL [Feldman & Rovner, 1969], and in the indexing structures of the PLANNER-related languages [Hewitt, 1972; Sussman, Charniak, Winograd, 1971; McDermott & Sussman, 1972], we can store away an arbitrarily large body of assertions and can easily retrieve any one of these later with a matching query. But the key word here is "matching": the datum to be found must be explicitly present, and it must be in the proper format for the match to succeed. These systems (ignoring, for a moment, their procedural components) give us no direct access to the much larger body of information that is *implicit* in the set of facts at hand. If we know things about "every elephant" or "every animal" or "every object bigger than a breadbox" and the questions are about Clyde, we need some way to connect the question to the answer. That means deduction, and deduction means search. To be sure that it has found all of the information relevant to Clyde, a knowledge-base system would have to examine a potentially very large set of related concepts.

The problem, then, is to find a way to perform this search in a reasonable time, even when the data base is huge. We can perform the deductions in antecedent fashion as new facts are added; we can perform them in consequent fashion in response to specific queries; or we can use some combination of these approaches, but the problem remains basically the same: our current search techniques are much too slow to handle a knowledge-base of sufficient size to produce a human-like intelligence, even in a restricted problem-domain.

Note that I am not referring here to the difficult deductions that people perform consciously: solving puzzles,

formulating hypotheses, deducing the voltage at some point in an electronic circuit, deciding whether some complex block-structure is stable, and so on. These, it seems to me, are legitimately the responsibility of the problem-solving parts of intelligence, and it is not too disturbing if they run rather slowly. It is the deductions that people find to be trivial and automatic, if indeed we notice them at all, that will concern us here. These are so tightly bound up with the operation and the contents of the knowledge-base that we must attack them as one problem. One of the major contributions of Charniak's thesis [1972] was to demonstrate just how much pre-existing knowledge comes into play in the understanding of a seemingly simple story intended for young children. It is significant that something like this had to be pointed out at all -- *this* is the kind of effortlessness that we must try to achieve in our machines.

The most general and mathematically elegant of the deductive systems, those based on some form of the predicate calculus, are ridiculously slow. The best of these, on the fastest computers, adopt a downright glacial pace when faced with more than a few dozen facts at a time. The PLANNER-style languages are somewhat better, since they give the user the ability to hand-craft, in the form of data-base demon programs, the exact deduction strategy to be used in dealing with each possible type of input or query. An optimal mix of antecendent and consequent reasoning can thus, in principle, be employed, and the searches can be guided to consider first those paths that are most likely to be productive. In actual practice, however, the principal advantage of such systems over the unguided deductions of the theorem provers is that the procedural systems do not have to be able to deduce every consequence of the knowledge at hand, but only those consequences that the system designer knows his programs are going to need. This makes the PLANNER-style knowledge-base an adequate tool for constructing the kind of limited test-system scaffolding that I mentioned earlier -- my own BUILD program [Fahlman, 1974a] is an example of such an application -- but it is still inadequate for implementing the sort

of knowledge-base that we will ultimately need.

The problem is that systems produced in this way tend to be very brittle. Only a carefully selected set of deductive paths has been implemented, so it is very easy for unanticipated queries or situations to cause the system to wander from these paths. At best, this lands the query back in the quagmire of undirected search; at worst, it causes outright failure. A solution to this is to build more and wider paths in an attempt to completely pave the area of interest, but this is paid for in vastly increased search-times. Even an optimized search, if it is to be reasonably complete, must sooner or later examine all of those concepts which might have something to say about the question at hand. If we are ever to endow our programs with something resembling common sense, we must somehow give them access not only to the most prominent and useful of an object's properties -- those that an optimized search would find first -- but also to those "fringe" properties that are usually insignificant but that may be of pivotal importance in a particular situation. An elephant's wrinkles are certainly well down in its list of prominent features, but to a tick they are home, and to a tick-remover they are the major obstacle to be overcome. Any single property of this sort may be used only infrequently, but the collection of them is so large that at any given time we are likely to be using some such property. The point is that we can rearrange the order of the paths to be searched to gain efficiency, but it is dangerous to leave anything out. We are left with a certain irreducible kernel of search to be performed, even if our strategies are very clever.

And who is going to write all of these search-optimizing programs? If these are to exploit the local regularities of the currently-existing knowledge and the local meta-knowledge about the likely patterns of knowledge-use, then the body of search-programs must be augmented and altered as the system learns new things. Unless there is to be constant human intervention in the system's inner workings, the computer itself is going to have to write these programs. Unfortunately, it is hard even for humans to write effective search-optimizing programs in

such a non-uniform environment. It requires not just technical skill but a good understanding of exactly how the knowledge-base is going to function. It seems unlikely that an automatic programming system will be able to exhibit such understanding any time in the near future -- in fact, it seems probable that to achieve such an understanding, the system would already have to contain the type of broad, flexible knowledge base that we are trying to develop here.

Despite these problems, the procedural approach to representing knowledge is still the dominant paradigm in the field of artificial intelligence (or at least that part of the field that has resisted the siren song of predicate calculus). Minsky, in his paper on frame-systems [1975], advocated the use of a combination of declarative information and local procedures to represent structured knowledge, and most of the workers that have followed Minsky's lead, notably Winograd [1974, 1975], Kuipers [1975], the FRL group at MIT [Goldstein & Roberts, 1977; Roberts & Goldstein, 1977], and the KRL group at Xerox-PARC and Stanford [Bobrow & Winograd, 1976], have continued in this vein. These researchers have been primarily concerned with the problems of flexibly representing relatively small bodies of knowledge, and of integrating the declarative components of their systems with the procedures. To the extent that they have addressed the problem of efficient search in a very large knowledge base, however, they have generally subscribed to the view that meta-knowledge, embedded in local search-guiding procedures, can eventually carve the searches down to a manageable size without destroying the generality of the system. This optimism may be justified, but to succeed by this route -- if indeed it is possible at all -- will require a tremendous investment of time and effort. In this report we will explore an approach that is much simpler and more direct.

1.3 The Parallel Network Approach

In my proposed knowledge-base system, we forget about trying to avoid or minimize the deductive search, and simply *do* it, employing a rather extreme form of parallelism to get the job done quickly. By "quickly" I mean that the search for most implicit properties and facts in this system will take only a few machine-cycles, and that the time required is essentially constant, regardless of how large the knowledge base might become. The representation of knowledge in this system is entirely declarative: the system's search procedures are very simple and they do not change as new knowledge is added. Of course, the knowledge base must contain descriptions of procedures for use by other parts of the system, including those parts that perform the more complex deductions, but this knowledge is not used by the knowledge base itself as it hunts for information and performs the simple deductions for which it is responsible.

The parallelism is to be achieved by storing the knowledge in a semantic network built from very simple hardware devices: *node units*, representing the concepts and entities in the knowledge-base, and *link units*, representing statements of the relationships between various nodes. (Actually, the more complex statements are represented by structures built from several nodes and links, but that need not concern us here.) These devices are able to propagate a variety of marker bits -- somewhere between 8 and 16 distinct markers seems to be the right number for human-like performance -- from node to node, in parallel through the network. This propagation is under the strict control of an external serial computer that is called the *network controller*. It is the propagation and interaction of the various marker-bits that actually constitute the deductive search.

The result is a network memory very similar to the one proposed by Quillian a decade ago [Quillian 1968, 1969], but with a very important difference: the network system I am proposing is much more tightly disciplined. The controller is not only able to specify, at every step of the propagation, exactly which types

of links are to pass which markers in which directions; it is also able to use the presence of one type of marker at a link to enable or inhibit the passage of other markers. It is the precision of such a system that gives it its power, but only if we can learn to use it properly.

Note that this is a very different kind of parallelism from that displayed by a relatively small set of serial machines working together. Ten CPUs, at best, speed up the processing by a factor of ten. Usually the improvement is much smaller because the CPUs begin to squabble over shared resources or because most problems cannot be broken up into ten independent, equal-sized parts. The proposed network, on the other hand, can perform many deductions in time proportional to the length of the longest branch of the search-tree, regardless of how many nodes the tree may contain overall. For short, bushy trees (knowledge bases consist mostly of short, bushy trees), the speed-up could be huge: a tree that is five or ten links deep might contain millions of nodes in its branches, so a million-fold speed increase is possible. Of course, this would mean that the parallel network must contain millions of hardware processing elements, but each element is very simple -- a few decoding gates, an internal state flip-flop, and enough other flip-flops to store the marker bits that may be present on that element. The knowledge itself is stored not inside the elements, but in the pattern of interconnections among them. (We will see later how this interconnection might be accomplished.) With current technology such a network would be very expensive, perhaps prohibitively so, but there is nothing mysterious or undefined about it.

The network scheme is not without its own problems, and we will examine these in detail, but the speed advantage in certain important areas is great enough to *qualitatively* alter our ideas about what is easy and what is hard. The network-based system easily performs many of the mental operations which seem effortless to people, but which have proven to be very costly (or very complicated) for serial computers: finding the implicit properties of an item in a large hierarchy of types and

sub-types (an "IS-A" hierarchy); dealing with multiple, overlapping contexts or world views; locating the known entity in the knowledge-base that best matches a list of distinguishing features; detecting any glaring inconsistencies between new knowledge and old; and so on. This rough correspondence of abilities does not necessarily imply that the human knowledge base uses a parallel network (though that is an interesting conjecture), but it does suggest that such networks might be *one* way to produce a system with human-like capabilities.

All of this talk about human abilities may cause some confusion as to my goals in this research. Let me state very clearly that this is meant to be artificial intelligence research, not psychology. People are able to do certain things with stored knowledge, and we want to find *some* way to make a machine do these things. The resulting theories may or may not prove to have some relevance to the human knowledge-handling system; it will take much careful experimentation to determine what the similarities and differences might be. It does seem to me that this general type of parallelism is *a priori* more plausible as a model for human knowledge-handling than systems which depend for success on the brute speed of current serial computers, since it is hard to see how the neurons of the brain could achieve such speed. Because of the speed advantage deriving from its parallelism, a network of the type I have been describing can achieve reasonably fast results even if the propagation time of its elements is in the millisecond range, instead of the microseconds or nanoseconds that we are accustomed to in our computers.

If my principal concern is not the modeling of human knowledge handling, why is there so much concern about what is hard and what is easy for people? Quite simply, I am using my own rather haphazard introspection in this area as a sort of heuristic guidance mechanism to tell me where to look and what to look for. If people seem to perform some task effortlessly, that is a pretty good indication that some efficient solution exists, though not necessarily on the kind of computer hardware that we are using at present; it is therefore worthwhile to

expend a certain amount of effort trying to find the processing strategy that makes the task so easy for the brain. If, on the other hand, both people and conventional computers have trouble with some task, it is possible that no good solution exists, and it is unlikely that the operation in question is an essential part of human-like intelligence; much less effort is indicated in such cases. Such intuitions, unreliable as they may be, are still a lot better than nothing. In the end, a theory in AI must stand or fall on its performance, the breadth and variety of the intelligent responses that it can produce or explain, and the extent to which it inspires better theories as its shortcomings become apparent -- not on the correctness of the psychological speculations that led to the theory.

One question should perhaps be dealt with before we go on: What is the value of a solution to the knowledge-base problem that is based on imaginary or impossibly-expensive hardware? I believe that there are four answers to this question. First, the expensive hardware of today may well be very inexpensive in the future. If we can clearly specify what we want and why we want it, the necessary technology is much more likely to come about. Second, there is the argument of pure science: the creation of useful systems is only a part of the goal of AI; equally important is the goal of understanding, in precise mechanistic terms, how the activities that make up intelligence can be accomplished, and how the time required by each method is related to the size of the knowledge-base. For this purpose, the expense or practicality of the hardware is irrelevant, as long as the system is well-defined. Third, there is the possible usefulness of this theory as a source of models and ideas for psychologists, linguists, and others concerned with the question of how the human mind functions. Finally, and in my view the most important consideration, there is the usefulness of the parallel network theory as a *metaphor*: an intellectual tool that will help us to factor out the constantly-distracting technical problem of search-efficiency from the more complex issues of how to represent and use the knowledge, given that the search is

accomplished somehow. Regardless of whether the deductive searches are ultimately performed by parallel hardware or by serial software, this separation of the problem will, I believe, help us to see more clearly the purely representational issues that we must deal with.

As I mentioned at the start of this report, the general idea of the parallel network system and the specific conventions and procedures of the NETL system are to a large degree independent. NETL has been specifically designed to run efficiently on the parallel network hardware, real or simulated, but it contains a number of ideas for improving the precision and representational power of knowledge-base systems in general, whether serial or parallel. Section 2 of this report will describe the parallel network system, along with its uses and general principles of operation; section 3 will cover the representational conventions and processing algorithms of NETL. A simulator for NETL is currently running in MACLISP on the PDP-10, and several test problems of assorted sizes have been run. These tests will be described in section 4, followed by overall conclusions in section 5. Appendix A will consider the possible hardware technologies for implementing the parallel network. Appendix B will summarize the node and link-types currently defined in NETL.