Working Paper 115

The Intersection Problem

by

Scott E. Fahlman

Massachusetts Institute of Technology
Artificial Intelligence Laboratory
November 1975

ABSTRACT

This paper is intended as a supplement to AI MEMO 331, "A System for Representing and Using Real-World Knowledge". It is an attempt to redefine and clarify what I now believe the central theme of the research to be. Briefly, I will present the following points:

1. The operation of set-intersection, performed upon large pre-existing sets, plays a pivotal role in the processes of intelligence.

2. Von Neumann machines intersect large sets very slowly. Attempts to avoid or speed up these intersections have obscured and distorted the other, non-intersection AI problems.

3. The parallel hardware system described in the earlier memo can be viewed as a conceptual tool for thinking about a world in which set-intersection of this sort is cheap. It thus divides many AI problems by factoring out all elements that arise solely due to set-intersection.

This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the laboratory's artificial intelligence research is provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-75-C-0643.

1. The Monkeys and the Beans: A Fable

There was once a tribe of monkeys who lived in a forest, not far from a small human settlement. One day a supply truck overturned on the rutted jungle trail and was abandoned along with its cargo: six hundered cases of canned baked beans. The monkeys, of course, were delighted with this inheritance; they had seen the humans eating these things, and the pictures on the cans looked very delectable indeed. The cans, however, were a problem. Soon the forest rang with the sound of cans being bashed against rocks, and rocks against cans.

After several hours of this, one of the younger monkeys became discouraged. "It shouldn't be this hard," he said. "Why is it so easy for the humans to get these things open? What are we doing wrong?"

The elders of the tribe smiled--they had been young and impatient once, themselves. "We cannot even begin to understand the things that humans do," said one, "and it is usually a mistake to try. It was before your time, of course, but the monkeys of the hill tribe used to spend a lot of time watching humans, trying to learn from them. They ended up wearing neckties! Some still wear them--at least in private."

"Rock-bashing is the obvious way to get anything open," explained the second elder. "Look how well it works for coconuts. And these" --he held up one of the cans-- "are really just metal coconuts. You can even hear sloshing sounds when you shake them."

"I know it's hard, slow work," said the third elder, "but we *are* making progress." He illustrated this by sucking a few beans out of a hole that had appeared in one severely battered can, smacking his lips with obvious relish.

The young monkey was not convinced. He disappeared for several hours. "I have been watching the humans," he announced upon his return. (The elders noted with relief that the youth's neck was still clad only in monkey fur.) "The humans have a little gadget that they use. It's called a can-opener, or something like that."

"Have you brought us one?" asked the first elder, with obvious suspicion.

"Well, no ..."

"Harrumph!" harrumphed the elders.

"... but perhaps if some of us made a diversion, the others could rush in and steal one. Or perhaps we could build one. There are some sharp metal pieces in the truck wreckage. Or perhaps ..."

But the others were no longer listening. Everyone had returned to the bashing.

After a while, a second young monkey spoke up. Though still a youth, this monkey was one of the strongest in the tribe, and had a fast-growing reputation as a champion coconut basher. "I have been thinking about this so-called can-opener," he announced, "and it is a bad idea. Am I correct in assuming that all it does is open cans--nothing else?"

"Yes," replied the first youth.

"Well, then, it's useless. Just getting the can open is really very easy. Observe!" With this, the strong youth raised a huge rock over his head and slammed it down on one of the cans. The can literally exploded, covering everything nearby with gooey baked beans and can-fragments. The trees, the rocks, the ground, the youths, the elders, all were coated with a glistening, lumpy, gelatinous brown. Wiping his mouth with a dripping paw, the young monkey resumed his lecture. "Can opening, itself, is trivial. The *interesting* part of the problem is gathering up all of the beans and separating them from the dirt, twigs, can fragments, and monkey fur. What possible use would this can-opener be in accomplishing *that* task?"

But the first youth was no longer listening. He was trying to get the beans out of his ears.

2. Hardware Is Not the Real Problem. Set-Intersection Is.

The ideas presented in my Thesis Progress Report [4] have given rise to a running argument within the lab. The issue has been (more or less) whether we really need some sort of parallel hardware to accomplish such vital AI tasks as symbol-mapping and recognition, or whether the standard Von Neumann machine can somehow be made to do the job.

The argument, in my opinion, is far from settled. Bob Moore [6] and Drew McDermott [5], in particular, have come up with indexing schemes for solving some of these problems, given certain assumptions about the form of the knowledge net and the conditions under which it will be accessed. These assumptions can be met for rather small problems, such as McDermott's electronic circuit designer. The question is whether they can be extended to much larger, less formal domains: McDermott thinks they can be; I have some doubts. Neither side of the debate has a clear enough picture of the overall problem of intelligence to prove its case, so the argument has degenerated into a battle of guesses. I am still inclined to believe that no Von Neumann machine algorithm will solve the more general cases with acceptable speed (whatever "acceptable" means in this context), but I am forced to admit that I have no way of proving such a thing.

Lately, however, I have begun to feel that this business about hardware is the wrong thing to be arguing about. It is, I think, merely the surface aspect of a group of much deeper issues, and a very distracting surface at that. The real issues concern the set-intersection operation: What role does it play in the processes of intelligence? How many of the problems faced by AI research are in fact caused by our inability to do set-intersection efficiently on our serial computers? How many other problems are the result of premature decisions about how to deal with certain awkward intersections? And how many are *real* AI problems, ones that we would have even if set-intersection were a trivial operation on the machines being used. On one level my parallel networks are a serious proposal in their own right, but on another they are just a device for studying issues such as these. The hardware gives us a precise way of thinking and talking about a world in which set-intersection is trivial.

It is important that we clearly understand what sort of set-intersection is being discussed here. Sets can be represented in many ways. They can be implicitly represented by generating-functions or predicates; they can be explicitly represented in the form of a list or a set of membership entries in a hash-table; or they can be represented in some intermediate form, such as the set of nodes present as leaves in a tree of transitive relations. My parallel network system can mark the intersection-set of any two other sets in essentially constant time, provided that these two sets are represented explicitly or as a fairly shallow tree. A few cycles are required to mark the set members, followed by a single cycle to actually do the intersection. This marked set is then ready for any subsequent operations; only if it is to be read out in serial must it actually be scanned. The key requirement is that the members of the incoming sets and the links that tie them together actually be present in the machine before the intersection takes place; if elements or links must be generated, the parallel system will be slow, and it is hard to conceive of any non-magic system that can do much better. All references to set-intersection in this paper, unless otherwise specified, refer to the intersecion of *pre-existing* sets.

Even if we confine ourselves to explicit sets, the serial machine proceeds much more slowly. A Von Neumann machine, in general, intersects sets in time proportional to the sum of the sizes of the sets. The system scans one set, marking each member or remembering the members in a hash table. Each member of the second set can then be tested at unit cost to see whether it is a member of the first. There is really no way to avoid scanning both sets if they are being seen for the first time; the only way to do better is to have some of the work done already, left over from previous operations on the same sets.

It would appear, then, that the Von Neumann machine is at a substantial disadvantage when faced with a task that requires very large sets to be intersected often. Do such tasks play an important role in the processes that we call intelligence? We must at least admit the possibility. It is certainly conceivable that the brain--the only working intelligence-machine that we know of--is built using some fast-intersecting parallel organization. If so, mental operations that look--and feel--trivial might actually involve dozens of intersections of huge sets, perhaps even in the million-member range. Researchers trying to duplicate such performance on slow-intersecting serial machines would have a real problem. In section 3 of this paper, I will present some examples suggesting that we are, indeed in this sort of position--that almost all data-base references and a great many more-complex operations of a truly intelligent system contain non-trivial set-intersections as essential parts.

Suppose this is true. Must we then abandon the Von Neumann machine as a tool of AI research? Not necessarily. One option is simply to stick to the clean solution--the general-purpose intersection technique--and to pay the price. Each major intersection may require a large number of cycles, depending on the size of the sets involved, but the fast cycle-time of the machine may make this acceptable in all but the most immense problem domains. The brain, after all, somehow gets along with "cycle times" apparently in the millisecond range, so the machine has a raw speed advantage of perhaps 10^4 or 10^5 to compensate it for any structural disadvantages.

Then, there are the various special-case tricks for using pre-stored results to speed things up. Recall that the general intersection technique involves scanning one set, noting its members, and then scanning the other set. Obviously, if we can afford the memory, we can keep the first set in its "noted" condition and perform any subsequent intersections by scanning only the other set. If both sets are pre-noted, only the shorter has to be scanned. For sets of equal size, this saves only about half of the work, but if one set is very large and the other is small, almost all of the usual work can be avoided. Remember, though, that this does not work if the larger set is being intersected for the first time.

If both sets are large, the only way to avoid scanning at least one of them is to know the answer (or at least part of the answer) in advance. The first time two sets are intersected, there is no alternative to doing the work, but if the intersection of the same two sets is ever needed again, the same answer can be returned. Of course, if these intersection-sets are stored indiscriminately, this procedure can eat up tremendous amounts of storage space. It is therefore important to recognize which of the intersections are likely to be repeated. If set-membership is ever allowed to change, the system has a formidable bookkeeping problem in keeping all of the stored intersection sets up to date. A variation of this technique is to save away the intersections of certain commonly-encountered subsets, but the constraints here can get so complicated that I won't go into all the gory details.

Now, it is entirely possible that some combination of these tricks can get the job done. Perhaps a way can be found to ensure that, in most cases, one of the sets being intersected is small. Where this is not possible, perhaps the same sets are being intersected over and over again, or are built out of subsets that are. Perhaps the remaining hard intersections can be avoided somehow, by adding more procedural cleverness to the system. But it is equally possible that there are many large intersections that just cannot be avoided and that must be done the hard way. If so, we will have no choice but to rely on a lot of pure, brute speed to solve the problem, or to turn to some sort of fast-intersecting parallel hardware. At present, nobody knows what the internal constraints of a total intelligence system really would look like. That is why the debate over hardware has become, for the moment, so vacuous.

The point of all this is to suggest that the standard approaches to AI, at least within the MIT clever-programming tradition, tend to obscure rather than illuminate these issues. Small, well-defined problems are chosen and are attacked using every trick in the book. Each case of intersection is dealt with separately, as it arises, by whatever combination of techniques seems locally appropriate. This inevitably introduces constraints and assumptions, and these are seldom fully understood and spelled out at the time. Later, when these unstated assumptions begin to bind and to clash with one another, it is very hard to locate the exact source of the trouble. Instead of being faced with a nice, straightforward case of sluggish intersection, we see complex problems of our own making, problems that are easy to confuse with real AI problems--those that arise from the nature of the tasks and not from our particular choice of machine architecture. A number of such pseudo-problems will be discussed in the next section. Of course, the really hard cases of intersection tend to collect in the spaces between the mini-worlds that people have chosen to work on. Little wonder, then, that linking up these islands has proven to be a difficult task.

My thesis research explores an alternative approach to all of this: *Assume* that we have some clean, general, low-cost solution to the set-intersection problem, and proceed from there. Many AI problems are neatly divided by this stratagem: first, we can explore the *real* AI issues using set-intersection as a trivial primitive operation; later, when we fully understand the dimensions and constraints involved, we can attack the purely technical problem of actually getting the necessary intersections done in reasonable time. Whether this is ultimately done by hardware or by software is an issue that need not--and *should* not--concern us at present. Perhaps the "pure" AI solutions will have to be modified a bit to accomodate certain intersection techniques, but by then we will be able to see the consequences of any such action. As I mentioned earlier, the proposed hardware is just a precise way of stating this assumption of fast-intersection and, perhaps, a psychological trick to eliminate the natural temptation of a programmer to cut a few corners for the sake of efficiency. Similarly, to study AI using a simulator for such hardware is just to constrain the programmer to always use the clean, general intersection technique instead of special-purpose tricks.

There are many possible ways to divide a problem-domain as large as AI. Some are useful; some make matters even worse than before. My initial reconnaissance of this proposed division convinces me that it is one of the good ones. Many of the really tough problems of AI--problems that were not anticipated in the optimistic days when the field was young--turn out to be well-disguised intersection problems. I have only just begun to sort out the non-intersection AI problems, but so far I have found nothing of comparable resilience.

3. An Inventory of Intersections

In this section we will examine a number of fundamental AI problems and operations, and will try to determine the role that set-intersection plays in each. We will also look at the techniques that have been used to deal with these intersections and the problems, if any, that these techniques have caused. This is just meant to be a quick survey, not a comprehensive analysis of the entire field or of any single case. The point is simply that non-trivial intersections occur far more frequently in AI than one might, at first, suspect.

3.1 Associative Data Bases

Let's begin with a problem that the current techniques handle rather well, just to get a feeling for how the game is played. That problem is the handling of pseudo-associative data-base items like (FATHER-OF CLYDE BERTHA). We are not (yet) including the complications that arise from multiple contexts or symbol-mapping. Clearly, to retrieve (FATHER-OF CLYDE ?) is in some sense to do an intersection: we want the intersection of the set of items with FATHER-OF in the first position and the set having CLYDE in the second. We can, of course, test membership in either set just by looking at an item, so we have only to scan the smaller of the two sets. Still, this might become tedious if we know both a lot of things about Clyde and a lot of father relations. Whenever we find ourselves in this position, we can simply store the intersection-set corresponding to (FATHER-OF CLYDE ?), and return that answer immediately if the question is ever asked again. That, really, is the basic idea behind most data-base indexing schemes: do the intersection by scanning the shorter list and, if that involves too much work, break up the lists by storing some or all of the intersection-sets directly. Of course, it takes a certain amount of bookkeeping to make it work.

There are many variations on the above theme, representing various choices in the trade-off of storage space, worst-case time, average time, overhead, flexibility of item-format, and other factors. One extreme approach is to pre-compute all of the intersections corresponding to questions that the data base is ever expected to be asked. The above example would be hashed into the following buckets:

(FATHER-OF CLYDE ?)	=> Who is Clyde the father of?
(FATHER-OF ? BERTHA)	=> Who is Bertha's father?
(FATHER-OF CLYDE BERTHA)	=> Is Clyde Bertha's father?
(FATHER-OF ? ?)	=> List all known father relations.

The other four possibilities would probably not be hashed, since patterns like (? ? BERTHA) do not correspond to generally useful questions. This technique ensures that the anticipated requests can be answered with a single hash-table fetch, at the cost of considerable redundant storage. It does not work very well for objects of arbitrary complexity, since there are too many possible question-formats that must be hashed, but for three-member items it works admirably.

In general, then, normal CONNIVER-type data-base references can be handled by one or another of these mechanisms with reasonable speed and with few unpleasant side-effects.

3.2 Multiple Packets

By now, everyone realizes that some sort of multiple data-base scheme is necessary for most AI applications--contexts, packets, or whatever. Let's consider the slightly more general packet system. Each data item is attached to (or "contained in") some specific packet. At any given time, some set of packets is said to be active. Each data-base request is really asking for the *intersection* of the set of matching items with the set of items residing in active packets. This intersection can be a bit more troublesome than the basic data-base fetch.

Clearly, if only one packet is active at a time, the packet-name can be simply appended to the item and treated as just another atom: instead of asking for (FATHER-OF CLYDE BERTHA), we ask for (FATHER-OF CLYDE BERTHA PACKET-42). All of the techniques of the last section apply.

Usually, though, the system must allow many packets to be active at once. In this case there are two possibilities. One is to scan the returned pattern-match bucket (or buckets if the pattern-matcher allows for variables to be present in the stored items) and, for each item, to see if it belongs to some member of the set of active packets. This discrimination can be made at unit cost for each item if the set of active packets is marked or remembered in a hash-table. The time is thus proportional to the size of the returned bucket. Note, however that each packet must be visited at least once whenever it is added or removed; it is no longer possible to switch on a large tree of nested packets in a single declaration. If we want to be really tricky, we can switch between several distinct sets of active packets, so that the set-up cost can be avoided if we return to an activation environment that has been used previously.

This technique of scanning the pattern-match buckets is more or less what CONNIVER uses for its context-system, except that in CONNIVER an item can reside in several packets instead of just one. Thus, in testing each item we must do yet another intersection to determine whether any of the item's packets are members of the active set.

Usually, though, the set of packets attached to a given item is small.

The other way of attacking this intersection is to access the data-base separately for each active packet, returning its own private bucket of matches for the requested pattern. I don't know of any system that uses this approach exclusively, but it is clearly preferable if the number of active packets is small compared to the size of the buckets involved.

McDermott's multiple-context indexing scheme [5] is an attempt to get the best of both worlds. If a match-bucket is small, it is left intact and is simply scanned. Once it exceeds a certain size-threshhold (but only after it is actually accessed in this bloated state), the system begins to break it up by packet-membership. It is not broken up all the way, however, since this would require scanning the entire set of active packets (or "cxt-layers", as they are called in this system) every time this pattern is requested. It is only broken down to a level which brings the buckets back under the size threshhold. For a given access-request, then, the system returns a tree of buckets. The terminal buckets must be scanned for items residing in active layers. If the context-tree is allowed to be tangled (or "spliced") there may be a number of these buckets, but at least each one is limited in size. The non-terminal buckets, one each for some fraction of the active packets, must be gathered and spliced into the answer, but it is not necessary to scan their contents.

This system is so complicated, and so strongly dependent on the exact form and history of the data base, that it is very hard to make general statements about how well it performs. In a few special cases it is able to eliminate almost all of the work, reducing very formidable intersections to a few trivial steps. (Of course, the work is done *once* when the tree of buckets is set up.) In other cases, equally special, this system does worse than the straightforward intersection methods, requiring steps corresponding to nearly every active layer and to nearly every item in the original match-bucket, not to mention the substantial overhead and set-up costs. But it is very hard to say where, between these two poles, the typical case for any given application falls without actually setting up the data-base and gathering statistics. I have played with a few mini-examples derived from sample IS-A trees (see McDermott's paper for the connection between IS-A trees and contexts), and it appears that the total number of tree-nodes and bucket-members visited is about a third of the number that would be required in simply crawling up the tree of active packets, but this is not really a fair test; for a large, fully-mature tree, in which there is enough room to employ all of McDermott's shortcuts and bypasses, the fraction of the total work saved would be considerably greater. A part of the savings is eaten up by overhead costs, but this is not significant where really big intersections are being done. Given our current state of ignorance about the exact form of these trees and the ways in which they will be accessed in practical applications, it is extremely unclear whether this indexing scheme will give us the orders-of-magnitude improvement in processing speed that we will need if really large intersections are to be rendered harmless.

Of course, none of this matters for most CONNIVER-style applications, in which only twenty or thirty packets are ever active at once, and the buckets are small anyway. But by concentrating on such applications, we become used to thinking of packet-filtering as a free operation, even though we know better if we stop to think about it. By keeping the real costs in mind, we will be less tempted to turn to packets as a free way of solving our problems with other large intersections.

3.3 Symbol-Mapping

It is clearly impossible for a system to store explicitly all--or even most--of the information that it will need to use. It seems very unlikely that for every vertebrate there is an assertion that it has hemoglobin, or that every elephant has its own private copy of the full-blown elephant description. Instead, we want to attach properties and relations to such abstract classes as "vertebrate" and "elephant", and to have these properties inherited by members of these classes and their sub-classes, sometimes many sub-class levels down the tree. Accomplishing this inheritance with reasonable speed is the essence of the so-called symbol-mapping problem.

Symbol-mapping, too, is basically an intersection problem. To find, say, the color of Clyde (an elephant), we must intersect the set of things that Clyde *is* with the set of things whose color is known. Clearly, the set of colored objects is too large to be scanned comfortably. The question, then, is whether the other set--the set of Clyde's superiors in the IS-A tree--is also large. If we are wedded to the Von Neumann machine, there is strong incentive to *assume* the smallness of this set, so that we can afford to scan it with almost every data-reference. Most of the proposals for practical symbol-mapping systems--Bob Moore's proposal [6], in particular--have included some form of this assumption.

There has been much discussion recently about whether or not we can afford to assume that Clyde, and all of the other objects in the world, belong to only a few property-bearing classes. I will not attempt to recount these arguments here. Let me just say that this seems to me to be a particularly good example of the kind of premature assumption that I was talking about earlier: one made to solve some immediate local problem, but whose long-range consequences cannot be clearly perceived. By incorporating such an assumption deeply into the structure of our research now, we may well be closing off all of the good solutions to some future problem or, what is worse, we might be making so much smoke and confusion that such future problems could not even be clearly identified. I think, in fact, that this assumption--tacitly made--has already had this effect in blinding us for so long to the possibilities inherent in large, tangled IS-A trees. These possibilities should be thoroughly explored before we resort to any massive and permanent defoliation of such trees.

Note that the problem is only disguised--not solved--by representing the superior classes as packets and the properties to be inherited as items in the packet. Where before we intersected the set of Clyde's superiors with the set of objects of known color, we must now intersect the set of active packets with the set of packets containing COLOR-OF assertions. Since, for this task, we activate exactly the set of packets corresponding one-to-one with the set of Clyde's superiors, the two approaches are computationally identical. The packet approach merely seems superior because we are so used to taking packet intersection for granted.

Of course, the approach does not have to be this tidy. We could, for instance, have the superior packets of *several* individuals active at once, making the intersection problem even more difficult and requiring yet another (much smaller) intersection to sort out the results. This, basically, is the solution that McDermott proposes for symbol-mapping [5]. He does add the suggestion that the results of any inheritance calculation be saved as an explicit fact to short-circuit subsequent calcualtions; this is certainly reasonable if we can afford the memory, but we are still faced with a substantial portion of first-time calculations. This trick can, of course, be used with any inheritance scheme. The burden of making this scheme work thus falls upon McDermott's context-indexing system; whether that system is enough to save the day for large IS-A trees is, as we have seen, an open question.

3.4 Recognition

Recognition problems arise not only in such I/O-related domains as vision and speech, but also in many types of problem-solving, where it is frequently necessary to recognize the similarities between the situation at hand and some problem whose solution is known. As I pointed out in my original thesis proposal [3], the recognition process can be divided into two phases: first, a hypothetical description is chosen from among the many possibilities, on the basis of a few matching features between it and the incoming sample; then, the hypothesis and the sample are compared and fitted to one another with much more care, to determine whether the hypothesis should be accepted, modified, or rejected. This comparison process is interesting, but it consists mostly of operations that are reasonable to do by conventional methods--it is, after all, not too different from the description-comparison that we see in the work of Winston [8]. It is the hypothesis-selection process that causes the real trouble.

Once again, the culprit is an intersection. The sample provides us with some set of perceived features; we must, if possible, locate some stored description that displays the same set of features (plus, of course, many others). To do this, we must *in effect* intersect the sets of descriptions that are associated with each of the incoming features. If we want to identify a six-foot-long green four-legged reptile in the Everglades, we must intersect the sets of six-foot-long things, green things, four-legged things, reptiles, and Everglades-inhabitants. None of these sets is so short that we would enjoy scanning it.

The situation is analogous to that encountered in retrieving items from an associative data-base, except that here the "items" have many more features, any number of which could be specified in a given match. This suggests a few possible approaches. One is to anticipate the exact set of features that will be seen for an object in the given item-class, and to hash the identification directly under that combination. Unfortunately, this requires that the incoming feature-set be anticipated *exactly*, and in any interesting real-world domain such anticipation is not possible. Consider how many different feature sets might come in for the class "alligator", if we include big ones, baby ones, those seen on land, half-submerged, hiding in the bushes, lying dead in the road, or drawn in some stylized form in a comic strip. Covering the field would require hundreds or thousands of feature-combinations to be hashed for this one class alone. Using this strategy without such combinatorial overkill would require that the low-level input processors produce a canonical description from a tremendously varied set of raw input presentations--an impossible demand except, perhaps, in the blocks-world.

The conventional indexing technique is more flexible, but is likely to involve considerable list-scanning, in addition to its high cost in redundant storage. The idea is to keep the item list associated with each feature intact until it reaches a certain size, then to begin saving away the intersections of this set with others. The system does not attempt to save all of the possible intersections, or even all of those that it has had to compute (many of these will never be seen again); instead, it saves selected intersection-lists, chosen on the basis of frequency of use and efficiency in carving up troublesome large sets. In the alligator example, we might end up scanning a list of large green reptiles or a list of Everglades reptiles, checking for the other features as we go. This is not an ideal situation, but is a very substantial improvement over the scanning of the raw feature-sets. Still more improvement can be obtained if the most common elements are pushed to the start of the lists, so that they will be seen early in the scan.

In general, this system will work fairly well as long as the right set of features comes in for an item: the expected ones, especially those that are unusual and thus cut down the set of possibilities very quickly. It will still get results if such critical features are missing, but only very slowly. The system is thus rather brittle, and great care and intelligence must be used in setting up the network of stored intersections. By way of comparison, the parallel network system is able to find an item from *any* set of properties that uniquely characterizes it, from the same network of properties that are used in data-accessing. Both approaches, of course, will have trouble if the set of given properties contains spurious features or features in some non-standard format, but at least with the fast-intersecting hardware we can check out proposed variations on the given feature-set at very low cost for each.

It may not be immediately apparent, but the sort of packet-based recognition system suggested in my thesis proposal [3], with its levels of recognition frames and suggestion demons, does the intersection of features in essentially the same way as the indexing system described above. The stored intersection-sets are given class-names and are represented by frames; the features which are intersected in to produce the next level of stored intersections become the triggering features for suggestion demons pointing to more specific class-frames. In terms of the computation that is done, it doesn't really matter that in one case the system is driven by an external search program and in the other it is driven by demons, since the action of the demons is so trivial: see a feature--suggest a frame. The frame system does provide for more graceful integration of special-purpose programs into the search process--programs, for instance, to ask the I/O system for certain verifications. It also may be more intuitive for human users. The price paid for this flexibility is an increase in overhead costs.

This central feature-intersection is so important in recognition that, if we let it, it will end up consuming 90% of our attention. This is too bad, because there are many other really difficult problems in recognition, and the last thing we need is to muddle these problems together with a lot of intersection-related concerns. Among these real recognition problems are dealing with spurious features, getting the features themselves into some more-or-less canonical form (perhaps by a lower level of recognition, perhaps with many levels running at once and communicating in interesting ways), processing exceptions, variable weights for different types of evidence, and the whole issue of detecting and describing the features in the first place. I have never held that by invoking the parallel hardware we are solving these issues; we are merely clearing away some of the intersection-related debris, so that we can begin to see what we are doing.

3.5 Class Monitoring

As AI programs go about their business, they must constantly be on the lookout for certain objects, events, actions, and generated goals that, whenever they are encountered, require some special action. An example for humans would be a rattlesnake: except in certain special situations (zoos), we want to do more than just recognize a rattlesnake and proceed about our business. Asimov's Three Laws of Robotics [1] would be another example, in that they prohibit the robot from certain broad classes of actions, and prescribe high-priority goals to be pursued in certain types of situations. Such monitoring operations have a lot to do with the phenomena that, in people, are referred to as "common sense" or "fringe consciousness" or "awareness". Each of the items to be looked for may be encountered only rarely, but there are a large number of them. Taken together, such monitoring operations play a large role in many kinds of processing. The problem, of course, is to be aware of these things when they appear, without paralyzing the system by constantly interrupting its processing to look for all of these items.

The usual solution is to use demons. For each item that we want to look for, we put a demon in the data base. We speak of these as though they were independent monitoring processes, but they do not really operate this way on a serial machine. In reality, a demon is a hash-table entry associating a certain pattern with the program specifying the action to be taken whenever that pattern is matched. Every time the system encounters a new object, proposes a new goal or action, or notices an event--in short, whenever it encounters some new entity that might be monitored--it looks in the hash-table under that object's pattern to determine whether a demon might be lurking there. The ongoing process must be constantly interrupted for this checking, but performance is not too badly degraded as long as each check is only a single, quick data-base reference. We must, however, be very careful not to let the routine check become a lengthy process.

Unfortunately, that is exactly what happens in most interesting real-world domains. The problem is that we want to monitor *classes* of items, but the monitor must be triggered by *particular members* of these classes. Asimov's First Law prohibits the harming of humans; it says nothing *explicitly* about sticking a knife into John Smith. Thus, to check each item, we must take the intersection of the set of classes that are being monitored and the set of classes that the item belongs to. To be absolutely safe, we would have to actually try to fit the item into each of the monitored classes and check for discrepancies, as we do in recognition. It appears, however, that human-level performance merely requires that the item be fitted into the IS-A tree (with a certain amount of digestion processing being performed) and that the set of monitored classes be compared with the item's *explicit* superior classes in the tree. This can still present a formidable intersection problem if the tree is large, and we have already discussed the difficulties that can arise if we arbitrarily assume the tree to be small. In fact, the relation between this type of monitoring and the use of simple demons is exactly parallel to the relation between property-inheritance and the use of a fully-expanded explicit data-base. Note that events and actions can have superiors in the IS-A tree, just like physical objects; see [4], section 19, for details.

Exclusive-set clash-detection is a form of class-monitoring. Every time we assign an object to one class, we prohibit it from belonging to a great many other classes. In effect, we want to set out demons that will look for any future assignment of that object, even indirectly, to one of these prohibited classes. In practice, every time a new assignment is made, the system must reassemble (mark) the set of prohibited classes for this object (it is hard to believe that the system would store explicitly, for each item, the set of things that it is *not*) and intersect this set with the object's new set of superior classes. Other forms of digestion can be driven by similar monitor-set intersections.

An interesting and closely related problem is described by Charniak [2] and, more recently, by Rieger [7]; I call it causal interpolation. We are presented with some situation from which a large number of possible goals and actions might be predicted. We also have some specific action which might be occurring for a number of possible reasons. The problem is to find the causal chain from the situation to the action, and thus to properly interpret what is going on. Rieger uses examples like the following:

"It started to rain. Ivan dived under the jeep."

"Ivan saw the precious water leaking from his radiator. Ivan dived under the jeep."

Clearly, in the first example Ivan is trying to stay dry, while in the second he is trying to catch the water or to plug the leak. The problem is how we are able to determine this with such apparent ease.

Upon examination, we find yet another intersection, or perhaps a series of them. We must intersect the set of action-types that can be predicted from the situation with the set of possible descriptions of the given action, or we must intersect the set of predicted goals with the set of possible reasons for the action. When faced with rain, people want to stay dry, farmers want to get the hay in, car drivers want their windshield wipers on, and so on. Staying dry might be accomplished by opening an umbrella, going or staying inside, or getting under something solid. This sort of information is presumably stored as part of the system's problem-solving knowledge, as possible techniques for accomplishing the "stay dry" goal. These sub-goals, in turn, have their own expansion-steps, resulting in a very large tree of potential action-types for this situation. The system would mark all of the actions in this tree (perhaps to some specified depth), and then would intersect this tree with the set of superior descriptions of the given action in the IS-A tree. Diving under a jeep is, among many other things, an instance of getting under something solid. Once the intersection process has found this essential action-type node, common to both sets, it can quickly verify that the causal path through this node does, indeed, make sense in the particular context at hand.

What is not totally clear, at present, is whether this intersection can proceed entirely through the network of existing, generalized action-types that must exist as part of the system's problem-solving knowledge, or whether the system will have to start generating new predicted actions and new descriptions for the given action. It is my belief that, in most cases, the existing network will contain the desired intersection-path, and that the causal chain can be found by the parallel hardware in a very short time. In a few cases, new elements will have to be generated, with a spectacular increase in the time and effort required, but this seems consistent with human performance--many actions are hard for us to understand because some of the hidden intermediate steps have never occurred to us and are therefore not an explicit part of our current knowledge-structure. Many details remain to be worked out, however, before I can demonstrate that the majority of these intersections are, indeed, of the static variety that my parallel networks can handle at high speed.

This concludes the survey. The list of ways that set-intersection comes up in AI goes on and on, but I believe that I have hit the most important cases.

One final thought: The set of situations which involve large, unavoidable intersections seems also to be the set of problems for which some AI researchers feel compelled to invoke communities of processors, if only as a figure of speech: demons, productions, actors, herds of conceptual herring gulls, or whatever. This is not surprising, since a truly fast and clean solution to the intersection problem must involve some sort of parallelism. It appears to me, however, that in many of these proposed systems the parallel elements are just smart enough to get into trouble. We see them arguing with each other, swallowing control and refusing to give it back, creating a Babel of interfacing conventions, and so on. My system, too, is a community, since each link and node is in some sense a processor, but these processors are much too stupid to cause any of these problems. I think this is a virtue. The proponents of the more complex systems might consider whether the complexity is really doing anything for them, or whether it is just a way of doing intersections; if the latter, perhaps the simpler system would be more appropriate.

BIBLIOGRAPHY

1. Asimov, Isaac, *I, Robot*, Fawcwett Crest (paperback), 1950.

2. Charniak, Eugene, "Toward a Model of Children's Story Comprehension", MIT AI TR-262, 1972.

3. Fahlman, Scott, "A Hypothesis-Frame System for Recognition Problems", MIT WP-57, 1973.

4. Fahlman, Scott, "A System for Representing and Using Real-World Knowledge", MIT AI MEMO-331, 1975.

5. McDermott, Drew, "Very Large Planner-Type Data Bases", MIT AI MEMO-339, 1975.

6. Moore, Robert, "A Serial Scheme for the Inheritance of Properties", SIGART Newsletter, August 1975.

7. Rieger, Chuck, "One System for Two Tasks: A Commonsense Algorithm Memory that Solves Problems and Comprehends Language", MIT AI MEMO-340 (forthcoming).

8. Winston, Patrick, "Learning Structural Descriptions from Examples", in Winston, *The Psychology of Computer Vision*, McGraw Hill, 1975.