NOTICE WARNING CONCERNING COPYRIGHT RESTRICTIONS:
The copyright law of the United States (title 17, U.S. Code) governs the making of photocopies or other reproductions of copyrighted material. Any copying of this document without permission of its author may be prohibited by law.

HOW TO VIEW THE COMPUTER

by

Allen Newell

DRC-15-1^-82

April, 19«2

How to View the Computer

Allen Newell
Carnegie-Mellon University, Pittsburgh, PA 15213

There is a certain amount of poetic justice in the present situation. It was Art Westerberg who came to me some months ago, proposing that I give this keynote address to you, a group of sophisticated chemical engineers involved in using computers to aid In the design process. He skillfully deflected each of my arguments about why I was not the right person to do this, until in the end I decided that, putting all notions of flattery aside, I was *indeed* the only person to launch this conference. Moreover, it would be intellectually stimulating for me to develop such an address. Now that, in the event, I am unnble to actually be with you, it Is Art himself who is to give these remarks - though I still have the fun and intellectual challenge of thinking them through.

I figure I am almost precisely half relevant to this conference, which Is one part computers, one part design and one part chemical engineering. I surely know about computers -- I get full credit there. I know a bit about design -1 assigned myself half credit on that. But of the most important part, chemical engineering, I know nothing at all.

Therefore, I must talk to you about fundamentals. For if one cannot talk about the details at hand -- and that is exactly my missing half -- then the choice is either fundamentals or irrelevaricies. By sticking to fundamentals, I shall not only seem profound (always a good thing in a keynote speech), but I shall be able to talk much about artificial Intelligence (AI) and computers, whereof I know something, and avoid chemical engineering, whereof I know little. Since you are all about to harangue each other for a week with intense chemical engineering talk, this is probably a good thing. The wonder is only that conference organizers locate keynote speeches nt the beginning, rather than in tlio middle, as breathers.

The most fundamental question that we have in common is how the computer is to be viewed. We all understand that we are on a moving train on an unknown track. That has been evident since the beginning - from the early fifties at least. The rate of progress has been sufficiently fast that all who have ridden the train for a few years have sensed perceptibly the changing nature of the computer under their intellectual feet.

As (at least moderately) hard headed engineers, you might wonder why viewpoints are Important. They are Important for the future. They govern the direction of progress -- how computers will develop to advance technology. Indeed, the very notion of using computers to aid In the design process is something that once was *pure* viewpoint, unencumbered by hard facts - a sense of pure potential. *

To a conference such as this, concerned with how to advance computer-aided design In chemical engineering, viewpoints are of the essence. Agreed, the meat and potatoes of such a conference always lies with direct extrapolation of current work, which rests within viewpoints already assimilated. But, if anything new is to happen, if a sense is to arise of new directions In which to move, then viewpoints will make all the difference. It is unlikely that anything an outsider like myself can say about viewpoints can be of service to you, but nevertheless thus *60*1 lustify my choice of keynote topic.

There has certainly been no dearth of viewpoints put forward on how to view the computer, either generally or within the scientific areas of concern to this group. These viewpoints range widely, from extolling the coming power, to cautioning that it is only man who thinks, *ne^mr* the computer. Let me state my main point right at the outset:

Existing viewpoints are of no service,

Now, of course, I cannot mean quite that, for I myself have a viewpoint that I think Is of modest service. Likewise, I do not simply mean to put down viewpoints that disagree with mine; I am too fair-minded for that. I wish to make a substantive point. Most of what passes for the conventional high-level technical wisdom In how to apply computers to the design process is in fact bad advice, given what we know currently.

I will discuss five viewpoints. I will prefer to state them as dichotomous issues, for various people have aligned themselves towards one *or* the other pole and I wish in general to argue

UNIVERSITY LIBRARIES
CARNEGIE-MELLON UNIVERSITY
PITTSBURGH, PENNSYLVANIA 15213

that the entire polarity is not ol service. Here are my five. I trust they are all familiar to you In one guise or another.

1. Heuristic computing ys Algorithmic computing - Or whether It to appropriate to use Al for practical computing.

2. Aiding man yjs poipq jl all -* Or that the computer should do what it does well and Man should do what he does well.

3. Task-oriented ya Engineer-oriented •• Or the engineer should help the computer vs the computer should help the engineer.

4. Procedural uniformity ya Complexity and ad tlQCJly - Or to use the computer efficiently implies finding simple and regular algorithms.

5. Computation, ya Knowledge - Or the computer is good for doing elaborate computations.

The central role of Al in these viewpoints should be evident. However, my concern to broader than that, covering the entire spectrum of how the computer shall be used. For I take it as axiomatic - a good term, if one is dealing with fundamentals - that how the computer to viewed, so shall it ultimately be used.

Without further ado, let us dig into our polarities.

HEURISTIC VS ALGORITHMIC COMPUTING

The Original View

The original view of the computer is enshrined in its name - a device to do computation. The model was the statistical clerk with a hand calculator. The universally agreed upon measure of program size was the number of multiplications. The distinction between a *heuristic program* and an *algorithm* was introduced in the mid fifties by the original workers In Al as a way to bring home that computers could be used to solve problems where it was not known exactly (or even very clearly) how to solve them, I must confess to being one of **the** original perpetrators.

The following general view **still** prevails. An engineer does an analysis of the task, making use of appropriate scientilic theories and mathematics, which result in a set of algorithms for solvino his problem, which is programmed for the computer. *This* is appropriate engineering nctiyily. On the conlrnry, gelling the computer to bohave intelligently is an exercise in not

using all that the analyst knows, but only providing some general common sense knowledge (indeed, the analyst could just solve the little puzzles himself, if he actually wanted them solved). Thus, although an Al program might be used for some ill-understood engineering task, as soon as sufficient engineering analysis is accomplished the Al program should be replaced *by a proper* engineering calculation.

I want to dissolve this distinction. I want to make clear there exists only a single issue: How to bring intelligence to bear on solving the final (here, engineering) problem. I will do this by first analyzing the nature of intelligent action and then examining the process of solving engineering problems by a set of intelligences distributed in time.

The Nature of Intelligence

It to fashionable to state that we don't know what intelligence is. In fact, good working definitions are available. Consider a situation (sketched crudely in Figure 1) with an *agent* who wants to perform some task, ie, to attain some *goal.* The agent has available some *actions {A l, A2,* ...) to modify the situation. He also has available some *knowledge;* ie, ho knows things about the task, about the relation of actions to effects, about methods, about the constraints of the situation - whatever. Then:

intelligence ia itifl abilily. to briofl knowledge to bear, on action to attain goals,

From a scientific viewpoint, the adequacy of this definition rests on the operationalization of *action, goal, knowledge* and *bringing knowledge to bear.* This is no mean feat. Yet, Al has made some notable progress on all of these terms. *Actions* are relatively non-problematical. An apparently satisfactory notion of *goal* *u>w exists, as a symbolic data structure that gives the knowledge of what is desired, along with associated information on potential methods, solving history, etc. Even *knowledge,* the toughest one of all, is pragmatically in quite good shape. Al has been able to formulate increasingly general global data bases, for which it has solved the problem of how to bring the encoded Knowledge *to boar* on data structures representing the current task situation, to select appropriate actions.

For instance, consider how *search* arises, which is usually taken to be the hallmark of Al programs. If the agent's knowledge is insufficient to select a correct next action (an external

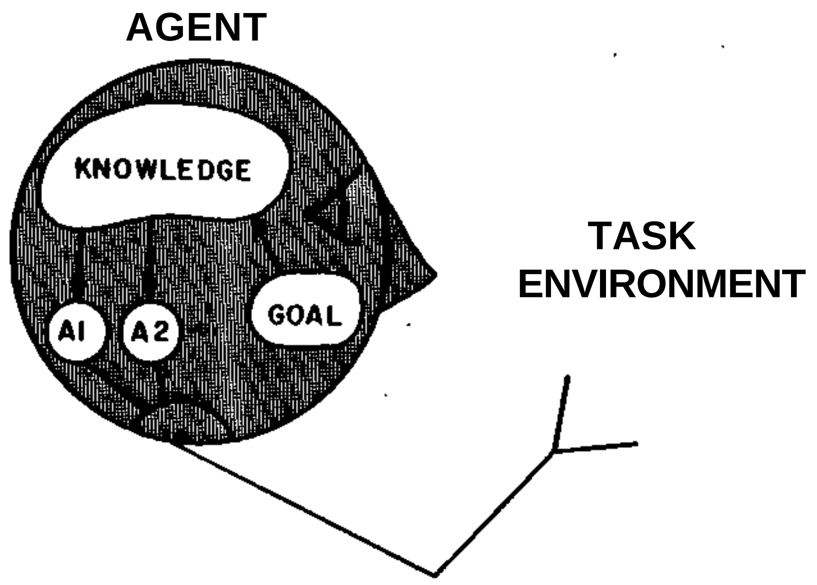

Figure 1: Intelligence

action In the completed task, an internal step in some subtask, whatever), then on occasion the action must be wrong, ie, not lead to successful task completion. When this error is recognized, a corrective action will be taken, and another positive step attempted. With this, a unit step in a search has lust occurred. Pervasive insufficiency of knowledge is the hallmark of the genuinely problematical. Hence, such errorful steps will cascade and -- voila - combinatorial search emerges. Thus, heuristic search arises naturally, given this definition of Intelligence.

Consider how routine behavior arises. In all cases, problematic or routine, the agent's situation Is *always* the same. At each instant he has a range of possible actions he can take towards goal attainment. However, as his knowledge increases, he (almost) always makes the right next action (and knows it is right, as well). Occasional failures are immediately recognized (Ie, more knowledge) and corrected without fuss (ie, still more knowledge), so they seem mere slips. His behavior seems to follow a definite procedural path, defined without reference to the possibilities of error and search. Thus, he moves into routine behavior.

But In fact he is always in a virtual search space. Those of you acquainted with D'Alembert's *Principle of Virtual Work* in mechanics will understand the right way to view the situation. Standard procedural programming languages make no reference to search and error, and this provides a model which makes routine behavior seem qualitatively different from Intelligent behavior. But this is one place where our current conceptualization of computers leads us astray. They are both of a piece.

Knowledge resides both in memories (as data structures) and in machinery (for doing things with Jhe data structures). The task of getting It out when needed is, of course, the whole art of being Intelligent. The knowledge can be there, but if It can be extracted only in *obvious* situations, the agent is not very intelligent.

Everything the agent has constitutes knowledge - facts about the task, procedures for bringing knowledge to bear, meta notions about his own operation, criteria for evaluating success, experience in solving past problems, etc. It is *all* knowledge: It all must be encoded; it all takes memory space; it can be used only by being brought to benr on some set of actions for some goals; and so on. This universal character is what makes this definition of intelligence theoretically pregnant. It also makes possible **agents that are universally** intelligent -• who can take on *any* task and proceed to work away at it (though not **necessarily** solve it).

Action at a Distance

In this view, when an intelligent agent solves a problem **matters are pretty** straightforward. The agent has some odd fund of knowledge, the situation is **encoded (using** some of that knowledge), machinations are gone through that bring some **of that knowledge** to bear (the more the better), and some solution occurs.

Now what happens with a program? It embodies a (perhaps limited) fund **of knowledge, a** problem is given to it, certain machinations are gone through, some **of that knowledge is** brought to bear and a solution occurs. A program is *in tact* an **intelligent agent. Thus, In** *writing* a program the programmer is creating an intelligent agent - **he is** creating **the fund of** knowledge etc., etc. That seems straightforward enough. Yet the **programmer** can only **endow** the program with knowledge that he **(the** programmer) has at creution time. **For he, himself, is** simply an intelligent agent, with a certain fund of knowledge, etc., etc. In particular, **at creation** time he is limited in his knowledge of the problem to be solved (by the program) **at task time,** which lies in his future. Thus, the programmer is creating intelligent *action at a distance,* **the** distance being mostly temporal, though also spatial on occasion. **The** situation **is sketched** roughly in Figure 2.

The programmer (equivalents here the engineer) **attempts to create programs that help** him solve his problems. What are the limits to what he can attain? Certainly **the engineer's** own intelligence is important - the ability to get his knowledge translated into **knowledge In** the program-agent. But the real limit is his inability to know the details **of** future **problems. The** ideal, of course, would be that whatever problems the engineer's **knowledge was capable of** solving, he could make that available at a distance. In fact, only in situations with highly precise specification of the future, can an extremely precise program be built **at design time.** This limit has nothing to do With the programmer's ability, only with the uncertainty **of the task** to bo performed.

If the engineer could select his tasks to be only those that were highly foreseeable in their

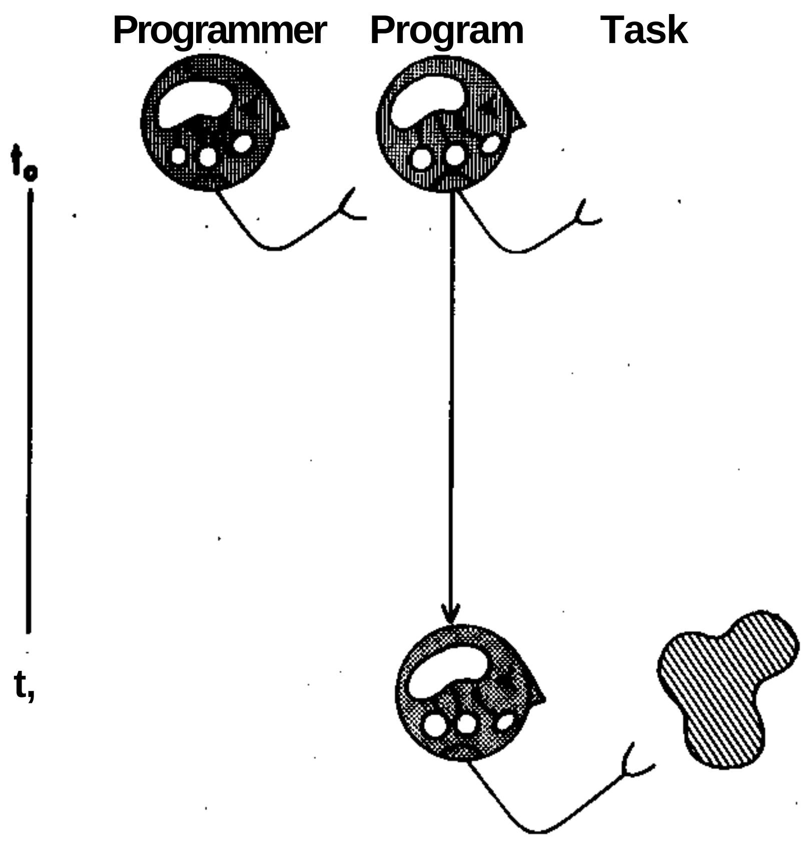

Figure 2: Action at a distance

structural detail, then he could write only programs that were correspondingly specific. But the engineer is not in control of this dimension of his task. All total tasks - for all time -- proceed from an inchoate start in some living COMIQXI, and pass through stages of sharpening, decomposition, restriction and proliferation. Suhproblems do get more well specified, partly from the acquisition of actual knowledge, but partly from stipulating facts *[Thou shaft adopt a linear programming model),* which just pushes uncertainty off to another place (to *the model justified?).* The full range of uncertainty will always exist. The game in writing programs will always be to package enough knowledge into the program to lump the gap between the engineer's uncertain knowledge about the problem at program-writing time, and the more complete knowledge that will be available to the program at program-operation time.

It is useful to apply this samo analysis to the engineer himsolf. For what happens In the four to seven years taken to educate on engineer? There is an act of building up a fund off knowledge at design time (ie, education time), to be applied to problems by the engineer at some later time. There are limits to the specificity of programming that is possible, precisely because the future set of tasks is not known -- and cannot be known - to the educators. Thus, educating an engineer can be equated functionally to writing a program. But If you think of the activity as *programming* you are doing yourself (and computers) a disservice. If you think of It as providing an agent with a fund of knowledge, etc. etc., then you are on the right track, both for education and for getting computers to solve problems.

Conclusion

By this little exercise. I want to impress on you that, relative to solving problems, we are all brothers under the skin - engineering programs, AI programs, and engineers. Relative to the task, we all package a fund of knowledge, which we bring to bear at task time. Getting intelligence to act at a distance is a groat thing, whether by education, instruction of humans, or programming computers. The more certain in Ihe future, the easier It Is to do. If the future were completely certain you coulil just solve it now, at design-time, and avoid programming (or education, etc.) altogether. No distinction of kind exists between algorithmic and heuristic* programs. There is only the attempt to <jet largei funds of knowledge to act fit a distance, so as to deal with more variable future; tasks.

AIDING MAN VS DOING IT ALL

The Original View

A theme that runs throughout the development of the computer is to find the *proper* distribution of tasks between the human and the computer. Here is a quotation from a just published book entitled *Sohware Psychology* (Sneiderman, 1980):

> "I argue that as time goes by, and even as we develop more sophisticated computer systems, the dichotomy between human creative skills and computer's tool-like nature will become more clear. ... To help lead us to (a) healthy ... position, which resolves the conflict by assigning tedious repetitive tasks to computers and reserving issues of creative judgment for humans, I propose five principles ...

(The principles are then enumerated, but they need not concern us.)

This concern Is not focussed just on the computer but spreads to all of the relations of man and machine. The field of Human Factors psychology can be said to be devoted to this wider topic. But the concern is even wider than that, as can be seen from works such as Lewis Mumford's *The Myth ol the Machine* (Mumlord, 1967), where the entire relation of man to his artifacts Is at Issue.

Though manifestly concerned with an engineering issue - the more efficient accomplishment of a total task - the underlying concern Is clearly moral. This is evident In the quote lust given. It can often be sensed in the implication that AI is not serving humanity properly -- that the *proper* use of the computer Is to *aid* man, not *replace* him.

The Moral and the Political

The moral issue Is certainly legitimate. We need to understand our relation to technology, especially as this technology becomes not only more pervasive, but more active and capable of autonomy. The computer Is certainly at the center of any such discussion. This moral dimension, of course, is linked Intimately with the *political* question of who controls whom, which always exists when there Is organized action to some end. Il there is a complex task to be performed and a collection of agents sets out to perform it, there will always be a division of work, with a consequent flow of signals which amount to commands and orders - agents controlling the behavior of other agents in various patterns of hierarchy, distribution and interaction. We recognize this situation in our everyday cultural institutions for dealing with governance and rights. It is surely complex and important, but it is not especially mysterious nor is it unfamiliar. It does not become more complex when the issue is between man and machine, as opposed to man and man.

Now, the point I would make is that these political issues exist independent of what functions occur in the division of work. They are questions of interests, rights and exploitations. When we look at the total task of design, in our current state of knowledge, we understand well how to use the computer for some functions, and not at all well how to use it for others. These happen to correlate with some notion of degree of intellectuality. But it makes no difference. If the computer did the total task, but did it only at our bidding, It would serve us. If it only computed energy balances, but did so at its own caprice and In its own interest, then we would have grounds for charges of insubordination. I conclude then that this dichotomy is not relevant to the issue of how to use the computer, but is primarily a smokescreen for this other, important but quite separate issue*.

TASK-ORIENTED VS DESIGNER-ORIENTED

Thft Original View

Back in the bad old days, computers were precious commodities. Almost any preparation off-line was deemed appropriate in order to minimize the use of the machine for dlspensible tasks to allow it to do what man clearly could not -- all those multiplications.

This same attitude still prevails. For instance, in speech recognition by computer it Is axiomatic that it must be *justified* to spend computer cycles on processing speech. Cost/benefit analyses are expected before research and development efforts are launched. Only ten years ago (1969), in a famous letter to the Journal of the Acoustical Society (Pierce, 1969), John Pierce of Bell Labs asserted that speech was not a *proper* input for computers.

> "When we look further for reasons [to recognize speech mechanically], we encounter that of communication with computers. ... In this general form, the reason is as specious as insisting that an automobile should respond to gee, haw, gkldap, whoa, and slaps or tugs of the reins. We communicate with children by words, coos, embraces, and slaps. ... It is not clear that we should resort to the same means with computers. In fact, we do very well with keyboards, cards, tapes and cuthode-ray tubes."

An Alternative History: The Increasing Capacity of the Computer

Yet, an Interesting history of the computer can be written in terms of increasingly spending computer cycles to help humans solve their housekeeping tasks of working with the computer. The development of higher level program languages is the first obvious example, but everyone could understand compilation as a computation-intensive task. Moving from batch to time sharing, is another such step - and one that was strongly opposed. The development of interactive editors, document production systems, message systems and the like -- all involve extending the computer to tasks that are not computation intensive (to a narrow analysis) but help the human with his unintellectual data processing tasks.

The one future fact we all know about the computer field is that the cost of computation - of cycles and of memory -• continues to decrease at a prodigious rate. I assume you all know this story. It Is currently heralded as the story of integrated circuitry, whose current acronym has reached VLSI (for *Very targe Scale Integration),* having progressed through IC *{Integrated Circuits),* MSI *[Medium Scale Integration),* and LSI *(Large Scale Integration),* and apparently leaving for the future only ULSI (for *Ultra* ...). A bit of historical perspective (over our ultra-short computer history) shows that there is always a current story for what is producing this remarkable growth, but the story changes from half-decade to half-decade. The computer field has been gaining at a prodigious rate for almost thirty years, and the only reasonable bet is that it will continue to do so, well after the integrated circuit development has had its half-decade or so.

The True Principle: Where Is the System Bottleneck

The relevance of the great advances in computer power is that cycles and memory will become freely available. Never *completely* freely available, but enough so that limitation of power cannot be used as the reason for not accomplishing a function by the computer.

The right principle, then, is that whatever tasks are limiting the productivity of the total system - those are the ones that we should be asking how to get on the computer. The focus should be on the total task, with no preconception of what type of function it is.

I have moved too fast. A caveat must be stated. The principle makes the implicit assumption that if only the function to be performed can be identified, then shifting it to the computer can be accomplished without further ado. But of course this is nonsense -- or at least it is *current* nonsense. The functions in question are any and all intellectual and information processing functions involved in total projects (such as the design and construction of new chemical engineering enterprises). Many of these functions are not well understood -- not scientifically, not computationally. We must qualify the principle above, since the key consideration may be our understanding of the function, not just its importance in the total task. But the limit is certainly *not* some apriori categorization of what belongs on the machine.

UNIFORM PROCEDURES VS COMPLEXITY AND AD HOCITY

The Original View

The fourth view I wish to discuss has rarely been an object of explicit controversy, but has simply been assumed. To wit: the natural, hence best, way to use the computer is with uniform and simple procedures. For instance, in some sampling I did of the chemical engineering literature, I found this statement by Sargent (1979):

> "It means that we now have the capability of solving realistic engineering problems by standard nonlinear programming algorithms, without the need for clever exploitation of particular features of the problem to make the computations practical."

Such a statement seems almost obvious. It fits with our notions of elegance and efficiency. It shades into a view that uniformity and well-honed standard algorithms are what computers do best. Yet, this view needs to be examined.

The issue has been argued considerably in AI, though focussed on the nature of intelligent processes, rather than the proper use of computers. This perspective is of use to us, so let me go into it a bit.

Procedural Uniformity

Practically since its beginnings, there has been a raging, but one-sided, debate in AI to the effect that uniform procedures are *bad*. It has been one-sided, because no one has really argued the contrary. It has raged, because AI programs turn out to be mostly quite uniform, occasioning the vehemence of the critics who feel the lesson has not yet been learned.

One example occurred in the early sixties. The original successes of AI occurred in tasks such as theorem proving in elementary logic and geometry, and elementary symbolic integration (Feigenbaum & Feldman, 1963). These programs used *heuristic search*, that is, sophisticated combinatorial trial and error. A reaction set in almost immediately, in which these programs (and the underlying research paradigm) were charged with not being concerned with knowledge and *expertise*. A quote from Joel Moses (McCorduck, 1979), a major figure in modern work in symbolic computation, gives the flavor:

> "The word you look for and you hardly ever see in the early AI literature is the word knowledge. They didn't believe you have to know anything, you [ie, the computer] could always rework it all."

Another (important) example occurred in the late sixties. Alan Robinson developed a form of first order predicate logic, called *Resolution* (Robinson, 1965), especially adapted for theorem proving on computers. Five golden years of intensive exploration followed. Resolution is a highly uniform procedure, built around a single inference rule (called *resolving*, naturally). First order logic can be seen as a sufficient base for general reasoning, and fair expectations bloomed. However, it soon became clear that resolution based theorem provers remained extremely limited (though much more powerful than their predecessors). In the early seventies this was converted to an argument about the inherent failure of uniform procedures.

Two things emerged from this. One was the development of a new family of AI languages (Planner, Conniver, QA4, ...) (Bobrow & Raphael, 1974) that attempted to provide the ability to write non-uniform programs, ie, programs containing much specialized and ad hoc knowledge. The other was a widespread conviction that uniform procedures had been demonstrated to be a bad thing for intelligent processes.

Insight into this debate can be obtained from my own personal experience. When Herb Simon, Cliff Shaw and I did the original LT theorem proving program (Newell, Shaw & Simon, 1957), we billed it as *complex information processing*. It was our view that we were dealing with supercomplex programs; indeed, LT was right at the limit of what we could cope with. Within a few years, LT came to appear -- to us and others -- as a simple program, whose complete structure could be captured in a single page of technical description. This suggests the limitation is in the program designer. His programs always turn out more uniform than he thinks, because the simplicity is essential for their initial discovery in operational form. The critics get no real argument, though no one seems to take their advice to produce *really* complex programs. It calls to mind the old catch-phrase, "I would if I could, but I can't so I won't"

Representational Uniformity

When thinking of uniformity, *procedural* uniformity comes first to mind - search and uniform logical inference in the cases jus! described. But that Is not the only type of uniformity. Representations can be uniform too. Jhe same considerations appear to prevail. The availability of uniform representations paves the way for progress in getting computers to perform intellectual functions.

One interesting case comes from work in chemistry that some of you may know, namely, that of the Dendral group at Stanford, which began in the mid sixties. It was the first successful effort to apply AI methods to serious scientific problems, using mass spectrogram data to determine the structural formula for complex molecules (Lindsay, Buchanan, Feigenbaum & Lederberg, 1980). It has been very successful indeed. The question I wish to raise is why did chemistry yield first?

Although any historical event always has many causes, a good case can be made for the critical role of the simple uniform representation inherent in the classical structural model of chemical molecules. It was there, available. The space of all isomers provided an arena within which additional processes could be defined and attached, such as heuristicalty guided search and evaluation functions linked to mass spectra. Indeed, the initiating step was a graph theoretic algorithm by Lederberg (called the *Dendral algorithm)* to generate graphs of all isomers without duplication, a step that already exploited the basic representation. The Dendral group was, to mix metaphors mightily, presented with a playground on a silver platter. Other areas of science have in general much more complex basic representations. So, of course, does chemistry, taken in toto. Significantly, however, a class of problems of genuine interest lay totally within this already invented, uniform representation.

yv second example lies closer to hand. In reading about process synthesis, I observed

that substantial progress has already been made in applying what are fundamentally AI techniques (Westerberg, 1079). Many of the first order things I could olfer you on applying AI techniques to design seemed already well understood and were being exploited in sophisticated ways. (This carried some weight, I suspect, in my decision that I must speak to you about fundamentals. No way was I going to discuss these programs without intensive study!)

However, though I congratulate you on your advanced state of computational sophistication, my real point may dull the luster a bit. For it seems to me that the advance has been made precisely because there was an available uniform representation, within whose closed world combinatorial problems of genuine applied interest could be defined. Take the central problem of heat exchange, which appears to be one of the furthest advanced. Conceptualization is possible in terms of a discrete set of streams plus a set of thermodynamic utilities, with the individual elements (the streams) characterized by a well understood small set of stated variables. Structure variation - the essence of why one cannot just cram such problems into an optimization format - is neatly combinatorial. It is no wonder you have been able to exploit what amounts to the basic heuristic search model of AI, discovering for yourself many fundamental lessons.

Conclusion

I am prepared to concede - even emphasize - the importance of uniformity. But in all these cases (and others as well) uniformity makes its difference in the ability of the designers to conceptualize what is going on. We humans are limited in our ability to deal with complexity. We can only create programs of a given degree of complexity relative to our own understanding at design time. Procedures and representations do not seem so complex after creation, when we become familiar with them. That is often noted. But then, importantly, we are ready to take another step in complexity in conceiving the next generation of programs. Throughout all this, the computer itself is prepared to be instructed and to perform according to any degree of complexity we can muster.

KNOWLEDGE VS COMPUTATION

The Original View

The final view I wish to take up is that computers ought to be used for *computation.* This again is not so much a point ol controversy, as an implicit assumption. Actually, our notion of computation has gradually expanded from nn essentially numerical conception (those multiplications, again) to include symbolic computation. Indeed, the chief Impact of At, when viewed critically, may not be so much intelligent processes, as the invention and routinlzatlon of manipulate symbolic representations. An interesting bit of evidence comes from the Dendral protect, already mentioned. Its major impact has been through a program called CONGEN, which is the manipulative guts of the total Dendral program that permits the generation and manipulation of i3omers and sets of, isomers. The intelligent processes were left to the AI researchers.

The computer is certainly already used for large data bases, and these are surely storehouses of knowledge, rather than computations. For instance, I noticed a strong emphasis in a recent work on process synthesis (Westerberg, Hutchison, Motard A Winter, 1979) on the data base of chemical constants. So my characterization, *computer* lor computation,* is not wholly accurate. Yet, such data bases, whether of chemistry data or Chem Abstracts, are somehow fundamentally *narrow.* They are not repositories of general knowledge about a domain. They are sets of numbers and class names that fit Into a world already parameterized.

Towards Handling General Knowledge

Recent advances in AI have moved us towards computer systems that do have mostly knowledge, and simply use that knowledge rather than engage in computation. These programs tend to be applied programs, and thus they are of special interest here. They are called *oxpart* programs - which actually is a pun. Recall the early controversial dichotomy in AI on search programs vr, expertise. Those reennt AI programs are supposed to be (at long last) the programs that have expertise in some task domain. Now, many of these programs turn out to embody the oxpurtirw of human ox ports in domains where the knowledge has

heretofore existed only in the expert's heads. These programs have required extracting this knowledge, and have become the only explicit form of this knowledge. Thus, they are also programs that embody the *human* expert's knowledge, hence they are properly *expert programs.* No conflict exists between these two senses; on the contrary, (he pun captures well the complex attitudes towards these programs in the AI field.

Many of these expert programs have been in medical diagnosis. For instance, MYCIN (Shortliffe, 1976), developed by Ted Shortliffe at Stanford Medical School and generally acknowledged as the program that set the style, provides consultation to a doctor about antibacterial agents. The doctor provides knowledge about his patient; MYCIN provides knowledge about bacterial infections, through a large set of rules that embody the knowledge of clinical experts. Other medical expert programs also exist (Pople, 1977, Weiss A Kullkowskl, 1979). However, the expert system I want to talk about, called R1, works in a quite different domain. It has just recently been developed by John McDermott of CMU (McDermotf. 1980).

R1 configures DEC VAX computer systems. That is, given an order from a customer, it analyzes the order to see that it is complete, adding additional components, such as power supplies, cables, etc, where necessary, and then lays out the components in the appropriate racks. You might think this is a trivial task, but that is not true. It is a genuine expert task. If you don't know anything about VAXs except that they are a modern computer, you won't be able to solve this problem *at all.* Of course, if you are one of the design engineers, it is all pretty obvious --)ust like bacterial infections to an expert clinician. On the other hand, if you are just an average employee, with only a modest amount of experience with VAXs, it is easy to make errors on this task. R1 is not a toy program. It is actually in use by DEC to process Incoming orders, Its expertise by now fully attested.

Figure 3 shows the first page of output from R1. The problem as input is on the right, under *Components Ordered.* The rest of the figure consists of RI's resulting output: the cabinets with the components that go in each cabinet. There are half a dozen more pages that give successively more detailed information about the components and their Interconnections, information of interest only to a customer and a manufacturer.

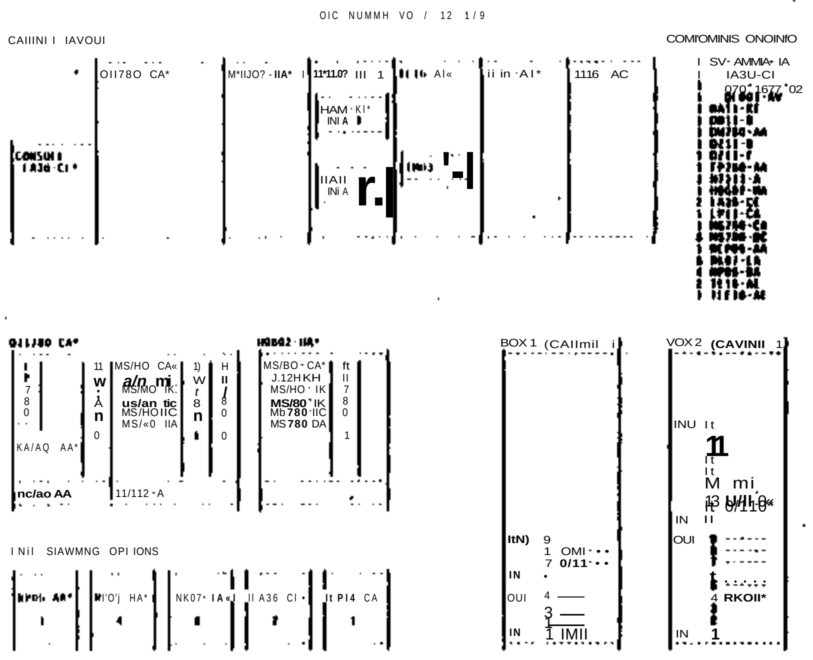

Figure 3: Task for R1.

The main reason for using R1 as an example is that its method is almost pure knowledge application. No *simple* test can be applied to a proposed final conliguration to tell if it is properly, configured, so no sort of trial and error works. Actually, that is not true. One can physically assemble the configuration and find out that it won't run - that is how errors are finally (and expensively) discovered. To configure VAXs requires knowing all the facts relevant to configuration about the 250 odd components that make up the VAX world. This knowledge is not uniform, but is of various kinds and has to be applied in particular ways that require still more knowledge. It is all rather straightforward for a human who learns about VAXs and spends some time configuring them. It is all rather novel for a computer, since there is no *computation* to do. R1 is not just a data base (though clearly it has one implicitly), for R1 itself applies its knowledge. It is my impression, in fact, that for quite awhile the inability to find what to compute stood in the way of using the computer to aid in this task.

To understand how R1 operates you have to know a bit about the programming system in which It is embodied. These are called *production systems,* taking the term not from manufacturing, but from the granddaddy system of this kind developed in 1038 by the logician Emil Post and called ever after *Post production systems.* In any event, as shown schematically in Figure 4, a production system consists of a set of rules, called *productions.* Each production has a *condition* part and an *action* part. All the productions look into a *Working Memory,* which holds the current state of the task and consists of a set of data structures.

The behavior of the production system is simplicity itself. A production whose condition is satisfied by the data in the Working Memory fires and its actions modify the Working Memory. This causes other rules to be satisfied, hence to fire, hence to modify Working Memory further. Many rules can be satisfied simultaneously, but only one fires at a time, principles of *conflict resolution* guaranteeing this to be the case. The behavior proceeds, then, simply by productions firing when they become relevant. No other forms of conditional action exist in this system. This system is about as pure an *apply knowledge* system as one can invent. It is, by the way and to put your mind at rest, a perfoctly general, powerful programming system, in which any computations whatsoever can be performed.

```
WORKING MEMORY:

      (description* dtscriptlonB . . . d«scr1pt1on»)


RULE MEMORY:

      (RUU-1: (situattonl «> actionl)
       RULE-2: (situation --> action2)
       RULE-3: (t1tuat1on3 --> act1on3)
          .
          .
          .
          .

       ASSIGN-POWF.R-SIIPPLY-2: (sUutttonl --> act1o.nl)
       ASSIGN-POWER-SUPPLY-3: (sUuationJ --> nctionj)
       ASSIGN-POWER-SUPPLY-4: (sUuationK --> action*)
          .
          .
          .
          .

       RULE-772: (s1tuat1on772 —> act1on772))
```

Figure 4: Production System architecture.

Rt is a production system. All of the knowledge about VAX components and how they are to be configured is embedded in a large number of productions. There are about 000 rules, of which some 500 are centrally concerned with configuring, the rest being concerned with housekeeping functions, mostly output. Here is a typical rule:

```
ASSIGN-POWER-SUPPLY-1

     IF: THE MOST CURRENT ACTIVE CONTEXT IS ASSIGNING A POWER SUPPLY
         ANO AN SBI MODULE OF ANY TYPE HAS BEEN PUT IN A CABINET
         AND THE POSITION IT OCCUPIES IN THE CABINET (ITS NEXUS) IS
             KNOWN
         ANO THERE IS SPACE AVAILABLE IN THE CABINET FOR A POWER SUPPLY
             FOR THAT NEXUS
         AND THERE IS NO AVAILABLE POWER SUPPLY
         ANO THE VOLTAGE AND FREQUENCY OF THE COMPONENTS ON THE ORDER IS
             KNOWN

   THEN: FINO A POWER SUPPLY OF THAT VOLTAGE AND FREQUENCY
         ANO ADO IT TO THE ORDER
```

This is an external format for the production, which suppresses some detail. The actual form is more algebraic. To do a typical configuration takes about 1000 firings of such rules.

Conclusion from R1

As I said, the beauty of R1 is that it is an almost pure knowledge program. It does its task simply by knowing enough about its domain to be able to recognize what knowledge is appropriate to apply when in the developing task situation. You should take from this that AI is learning to get computers to use diverse knowledge, and not fust to do heuristic search (surely a form of computation).

CONCLUSION

We have now worked our way through the five Issues. Each has provided a polar pair of viewpoints on a different aspect of the fundamental nature of the computer. All the issues bear on how we shall use the computer. Some have the flavor of *should,* others only of *obvious* or *efficient.* Let us recap these viewpoints along with the specific attitude I have tried to engender about each. For this talk has been no neutral, analytical exercise in examining these viewpoints. I have pushed particular conclusions In each case. So, here are the polar issues plus my own opinions.

The VlewDOlnta and their Upshot

The first issue was *Heuristic vs Algorithmic computing,* in which the received view in the engineering community Is that engineering computations are properly algorithmic. By examining the fundamental nature of Intelligence and programming, I tried to show there Is no distinction. AH Intelligence Is bringing knowledge to bear, all programming is trying to do that at a temporal distance.

The second Issue was *Aiding man vs Doing it all,* In which (again) the received view is that the computer should operate as an aid to man and not be used to completely automate tasks. I tried to show that the driving force behind this polarity was moral and political. These Issues are Important and may even dominate on occasion. But they should not be confused with what functions we get computers to perform.

The third Issue was *Task-oriented vs Engineer-oriented,* where the received - but fading - view Is that the computer Is Ill-employed if Its cycles are devoted to helping the designer with the Informational trivia of his task. On the contrary, the only criteria are what limits the total system and whether we understand the intellectual function.

The fourth issue was *Uniformity vs Complexity and ad hocity,* in which the received view is that computers are good for uniform procedures. *I* agreed that uniformity helps in applying computers, even showing that representational uniformity was important, as well as procedural uniformity. However, I argued that uniformity is primarily an aid to keeping things simple for designers in creating programs. It does not aid the computer at all.

The fifth and last issue was *Knowledge v$ Computation,* where the received view Is that computers are to be used for computation. The difficulty here is not so much a notion of what is *proper,* as one of limited understanding. Given recent progress In AI, I was able to make the point substantivoly that computers are as apt for knowing as for computing.

The True Viewpoint

So, what should be the *true* viewpoint towards that computer? It comes in several forms, some a little flip, some more reasoned.

First version; None of the existing views are serviceable.

This is the one we started with, that led me to use my time beating down existing views. It It primarily negative in tone.

The second form can be stated (with due apologies to Dante):

Second version; Abandon cliches all ye who eqter here.

The upshot of all of my argumentation has been that no simple classification can describe what the computer is good for or how it ought to be used. This is still a negative conclusion, but contains the seeds of an explanation.

A final, more positive, version is:

Third version: M intellectual functions aia natural 1Q Ihfl computer; the limits tQ aopJicaiioo !i£ in our understanding of ilifiss functions.

This viewpoint rests on the bedrock of the continued exponential growth of computer power (memory, speed and cost/performance). One can no longer rely on the instinctive engineering reaction to look for the limiting factors in the cost structure of the machine. In the instant short run, of course, such limits will exist (I can't get enough computer cycles, either). But this conference is concerned with how to advance computer-aided design of chemical engineering systems. It operates with a time horizon of five to ten years. New conceptual

paths launched now, at this conference, will reach fruition in that order of time. Five to ten years Is *long* compared to the rate of growth of computing power. The computing power constraint is no longer of service in structuring our view of the computer.

The real limitation is that one cannot (yet) get the computer to accomplish novel intellectual functions where we ourselves are lacking scientific understanding of these functions. Manifestly, there is much we do not understand. Several examples were evident during the talk. However, the time scale on changing our understanding is also of this same order, five to ten years. If certain intellectual functions have a high payoff for design, that they are not well understood *now* cannot suffice to dismiss them. The research you initiate could well include understanding intellectual functions not now well in hand.

Finia

So much for fundamentals. I hope it has gotten just close enough to the mark of your technical concerns with design so that during the conference you will catch yoursolf using one of these five viewpoints in an argument or discussion about what can or should be done in computer aided design of chemical engineering systems. With luck, you may be caused to think twice about what is appropriate - and to stretch your view a little further.

I say again that I am sorry that I have not been able to be with you on this occasion. Thank you, Art, for giving voice to my thoughts. Good wishes to you all for the success of the conference.

LITERATURE CITED

Bobrow, D.and B.Raphael, "New programming language for AI research," in *Computing Surveys,* 6, pp. 153-174 (1974).

Feigenbaum, E. and J. Feldman (Eds.), *Computers and Thought,* McGraw-Hill, New York (1063). (Reprints most of the important early heuristic search programs.)

Lindsay, R. K., B. G. Buchanan, E. A. Feigenbaum and J. Lederberg, *Applications ol Artificial Intelligence to Chemistry: The OENDRAL Project,* McGraw-Hill, Now York (1060).

McCorduck, P., *Machines Who Think,* Freeman, San Francisco (1970). (Moses is quoted on p. 228.)

McDermott, J., *R1: A Rule based Configurer of Computer Systems,* Technical Report, Computer Science Department, Carnegie-Mellon University (1080).

Mumford, L, *The Myth ol the Machine,* Harcourt, Brace Jovanovich, **New York (1967).**

Newell, A., J. C. Shaw and H. A. Simon, "Empirical **explorations of the logic theory machine: A case** study in heuristics", in *Proceedings of the 1957 Western Joint Computer Conlerence,* Western Joint Computer Conference (1957).

Pierce, J. R., "Whither speech recognition," *Journal of the Acoustical Society of America,* 46, pp. 1049-1051 (1969).

Pople, H. E. Jr., "The formation of composite hypotheses in diagnostic **problem solving: An** exercise in synthetic reasoning," in *Proceedings International Joint Conference on Artificial Intelligence,* MIT (1977).

Robinson, A., "A machine-oriented logic based on the resolution principle," *Journal of the Association tor Computing Machinery,* 12, pp. **23-41 (1965).**

Sargent, R. W. H., "A review of optimization methods **for nonlinear problems," in *Proceedings*** *Computer Applications to Chemical Engineering Process Design and Simulation,* **IAEC** Division, **ACS** (1979).

Shortliffe, E. H., *Computer-Rased Medical Consultations: MYCIN,* **Elsevier, New York (1976).**

Sneiderman, B., *Software Psychology,* Winthrop, Cambridge, MA **(1980). (Quote from p. 272).**

Weiss, S. M. and C. A. Kulikowski, "EXPERT: A system **for developing consultation models,"** in *Proceedings International Joint Conlerence on Artificial Intelligence,* **MIT (1979).**

Wosterberg, A.W., H.P. Hutchison, R.L. Motard **and P.Winter,** *Process Ftowsheeting,* Cambridge University Press, Cambridge, MA (1979).

Westerberg, A. W., "A Review of Process Synthesis," **Carnegie-Mellon University (1979).**