Planning Routine Computing Tasks: Understanding What to Do

by

Suzanne M. Mannes and Walter Kintsch

Institute of Cognitive Science

University of Colorado
Campus Box 345
Boulder, CO 80309-0345

ICS Technical Report #89-9

Planning Routine Computing Tasks: Understanding What to Do

Suzanne M. Mannes & Walter Kintsch[1]
University of Colorado

1. Problem Solving and Comprehension

Problem solving is the quintessential cognitive activity. But there is probably no single, unitary psychological process of problem solving. There are many different problems to be solved and many different ways to solve a single problem. A dimension along which these ways can vary ranges from apparently fixed, precompiled, highly automated procedures to flexible, compositional procedures, involving a great deal of deliberation. These psychologically different processes may contribute to different degrees and even interact in the same problem solving episode, as when a chess expert analyzes precompiled moves which have been retrieved via pattern matches.

On the one extreme there are fixed procedures which are run off more or less automatically in familiar situations. Highly overlearned action sequences like starting a car belong to this class. Colloquially, one does not even use the term problem solving in cases like this, although a person unfamiliar with cars would have to do a certain amount of problem solving in this situation. At the other extreme, there are problems like designing a kitchen, requiring a great deal of time and conscious effort, planning, goal and subgoal formation, and evaluation. This is the kind of behavior typically thought of as real problem solving.

1.1 The Construction-Integration Model

There is, however, a class of behaviors that fall in between these two extremes, where fixed, precompiled responses no longer suffice, but which do not the have the deliberate character of real problem

---

[1]This research was supported by the Army Research Institute, ARI Project MDA903-86-CO143. The work reported here was part of the PhD dissertation of the first author. We gratefully acknowledge the help and advice received from the members of the dissertation committee, especially Clayton Lewis, Peter Polson, and Gerhard Fischer.

solving. Colloquially, these processes are often said to involve comprehension. Text comprehension provides a good example: subjectively, reading an easy text is more akin to perception than problem solving. We don't think much, we just read along, as long as all goes well. Of course, when we get stuck, some real problem solving is done, unless we choose to just disregard the difficulty and go on with the easy reading. As has been argued in Kintsch (1988), most theories of text comprehension have not done justice to this phenomenal feature of comprehension processes. Rather, they liken comprehension to deliberate problem solving. The model of comprehension proposed by Kintsch (1988), on the other hand, breaks with this tradition. It is a computational model of text comprehension, describing the construction of a mental representation of a text with simple, though rough and crude rules as a bottom-up, data driven, highly automated process that requires little conscious control. Crude rules create a crude representation, however. Fine-tuning is therefore necessary, which can be achieved through a wholistic integration process in the connectionist manner: elements of the representation that fit together strengthen each other, and reject irrelevant or contradictory parts. Comprehension is thus seen as consisting of a construction phase, where quick and dirty, imprecise rules are used promiscuously, followed by an integration phase in which a coherent picture emerges from what had been carelessly put together before.

A good illustration of how sophisticated understanding can be achieved in this way is given by the application of the construction-integration model to word problems. Young children solving simple arithmetic word problems experience a great deal of difficulty and make highly characteristic errors which can be used to infer their cognitive processes (Cummins, Kintsch, Reusser, & Weimer, 1988). Their behavior can be modelled in two ways: as a schema-driven highly sophisticated inferencing system employing complex, smart rules (Kintsch & Greeno, 1985), or in terms of the construction-integration model (Kintsch, 1988). In the latter, instead of trying to come up with exactly the right arithmetic interpretation for a given problem, all plausible hypotheses are formed in parallel, and the one is chosen that is supported best, even if not fully, by the context provided by the text of the problem. This approach has certain advantages over the powerful-rule alternative in that it accounts for some empirical phenomena which are difficult to handle otherwise. It is also attractively simple computationally.

It is this type of model that we propose to extend here to a different type of problem solving, the domain of routine computing tasks. The basic assumption is that you don't try to figure out what to do when performing such a task, but let the textual instructions and the dynamic situational context select what to do from a broad range of alternatives. Thus, problem solving in the word arithmetic domain is seen as involving the contextual selection of one of a number of automatically activated arithmetic hypotheses, and problem solving in the routine computing domain as involving the contextual selection of one of a number of possible commands.

1.2 Associative Networks

A central assumption of the model of discourse processing proposed by Kintsch (1988) and extended here, is that of human memory as an associative knowledge base. Associative networks have a long history, but they have been overshadowed in recent years in both psychology and artificial intelligence by more structured knowledge representations, such as semantic networks, frames, and schemata (e.g., Collins & Loftus, 1975; Minsky, 1975; Schank & Abelson, 1977). Although the structure of a schema provides great advantages when it comes to the control of cognitive or simulation processing, the same structure can easily turn into a straightjacket when it cannot be adjusted with sufficient flexibility to an ever changing environment. Associative networks, without fixed global structures, offer a promising alternative in this respect, as long as it is possible to generate global structures from the local information contained in the network in response to particular task demands. In this way, at least in principle, the generated structure would always be contextually appropriate, because it was derived in response to the constraints imposed by that context. There is no need to modify a schema that never quite fits the specific context. Rather, a schema is created on the spot, suitable for that specific context. That such contextual generation of global schemata out of the local relations in an associative network is possible under some special conditions has been shown by Rumelhart, Smolensky, McClelland, & Hinton (1986) and Kintsch & Mannes (1987). The work reported below extends this approach in new directions.

1.3 Routine Computing Tasks

Routine computing tasks (i.e. tasks which are done in a semi-automatic manner on a regular basis), for expert users, belong to the

intermediate class of problems discussed at the beginning of this paper. They cannot be solved with fixed scripts, because although each component of the task is highly familiar and script like, the whole sequence that needs to be performed may never have happened in quite the same way before. Simple actions like displaying a directory, or editing a file are not at issue here; how to do that can be represented by fixed, stable, decontextualized knowledge elements. In the tasks considered here, these knowledge elements have to be put together in ever new sequences and in ever new variations. Some of these sequences might recur frequently enough and, as a result, be compiled into scripts themselves, but at present these exceptions are neglected. Instead, the focus of the present paper is on the more usual case where partly or entirely new action sequences must be generated to perform the assigned task.

Traditional psychological theory and artificial intelligence programs can deal with tasks at both ends of the composition continuum. Many of these works carry out effortful problem solving, as characterized by GPS (Ernst & Newell, 1969) which solves game type problems with which humans find difficulty and MOLGEN (Stefik, 1981) which designs molecular genetics experiments requiring too many constraints for a human processor to keep track of effectively. Alternatively, artificial intelligence has also produced programs which produce aspects of cognition which people exhibit when they solve a problem effortlessly, as in swapping the top blocks of two blocks world towers (e.g., Sacerdoti, 1977).

Here an alternative is proposed, in terms of the kind of middle-of the road, comprehension-based account referred to above. This is done for two reasons: deliberate problem solving is hard work, under conscious control - but in protocols which subjects performing routine computing tasks have provided there is little evidence of that. Subjects seem to just understand what they are supposed to do. Secondly, the exploration of how a theory of discourse comprehension can deal with these tasks that, at a descriptive level, obviously involve solving problems, is desireable. Where are the boundaries between the domains of perception, comprehension, and problem solving? We do not present here any empirical results that could sharply discriminate between the comprehension model proposed here and more traditional problem solving explanations. Our goal is merely to establish the feasibility of our approach and to explore its implications.

In the next section an outline of the NETWORK simulation is presented. It is an implementation of the construction-integration model for the routine computing domain. The subsequent section describes how this system understands and plans to solve some prototypical tasks. Finally, it is shown how case-based reasoning can be incorporated into the system. These topics are further discussed in Mannes (1989).

2. The Simulation: NETWORK

The goal of the simulation is to understand brief texts that request a computer user to perform certain simple, routine computing tasks. Understanding here means, inter alia, knowing what to do. More specifically, the simulation must compose a series of actions that, if executed, perform what was requested. The simulation knows about a set of basic actions - called plan elements - and it attempts to string together the right ones as needed - i.e., it forms a plan to perform the task.

NETWORK has two components, a general knowledge base and procedures for understanding specific tasks. The knowledge base about how computer systems work and what routine tasks involve is used in understanding the instructions to perform specific tasks. This knowledge base is referred to as the long-term memory of the system. Given a specific instruction text, specifying a state of the world and a task to be performed, NETWORK uses its procedures to form a task representation which consists of a representation of the text itself plus certain knowledge enrichments, including the aforementioned set of plan elements. Following this construction phase, an integration process activates the plan elements differentially, depending upon the strength of their connections with the rest of the text and each other. The simulation executes the most highly activated plan element whose conditions are met in the world. This action changes the state of the world, and the whole process recycles, executing another plan element, until the desired outcome has been achieved.

2.1 The Long-term Memory Network

Long-term memory is conceptualized as an associative network. The nodes of this network are propositions (or concepts, which can be regarded as a type of proposition). Propositions were chosen as the basic unit for NETWORK because they allowed information about the

domain to be distributed in the system's memory and because of the psychological validity of propositions as the basic processing unit in text comprehension (Goetz, Anderson, & Schallert, 1981; Kintsch & Keenan, 1973). Nodes representing concepts that are in some way related - because of spatio-temporal associations, or because they are intrinsically related (e.g. semantically or causally) - are linked together, some links being strong, some weak, and many zero.

Our simulation of this complex and intricate associative network is rather crude and perfunctory. We do not do justice to either the magnitude or complexity of human memory. Instead, only a small set of nodes are used which are more or less directly relevant to the tasks to be performed, and these are linked in a way that approximates human memory organization. It would certainly be desirable to work with a more sophisticated knowledge base, but that would be a major project in itself at the present level of understanding of the issues involved, and it is not the focus here. For present purposes, this simplified approach to simulating long-term memory suffices, if just barely so.

2.1.1 Protocol Analyses

To obtain the nodes for the simulated long-term memory network, a protocol study was performed in which 6 experienced computer users were given several routine computing tasks to perform while providing concurrent think-aloud protocols. All tasks involved routine actions such as handling and editing files and using the mail system, and the tasks that will be discussed in Section 3, where they are used to illustrate the present model, are among these. The verbal protocols were propositionalized (according to the standards of Bovair & Kieras, 1985 and following Kintsch, 1974) and used to form the basis for the long-term memory network. They represent the core of what our simulation knows about performing these tasks and can be categorized as two types of information; general knowledge about computers and the tasks to be done, and knowledge about plan elements which are commands to execute in order to accomplish the tasks.

2.1.2 Enrichments to Protocol Contents

This core of long-term memory contents was enriched in three ways. First, six different subjects were asked to provide free associations to the original nodes in the network (i.e. the original propositions

domain to be distributed in the system's memory and because of the psychological validity of propositions as the basic processing unit in text comprehension (Goetz, Anderson, & Schallert, 1981; Kintsch & Keenan, 1973). Nodes representing concepts that are in some way related - because of spatio-temporal associations, or because they are intrinsically related (e.g. semantically or causally) - are linked together, some links being strong, some weak, and many zero.

Our simulation of this complex and intricate associative network is rather crude and perfunctory. We do not do justice to either the magnitude or complexity of human memory. Instead, only a small set of nodes are used which are more or less directly relevant to the tasks to be performed, and these are linked in a way that approximates human memory organization. It would certainly be desirable to work with a more sophisticated knowledge base, but that would be a major project in itself at the present level of understanding of the issues involved, and it is not the focus here. For present purposes, this simplified approach to simulating long-term memory suffices, if just barely so.

2.1.1 Protocol Analyses

To obtain the nodes for the simulated long-term memory network, a protocol study was performed in which 6 experienced computer users were given several routine computing tasks to perform while providing concurrent think-aloud protocols. All tasks involved routine actions such as handling and editing files and using the mail system, and the tasks that will be discussed in Section 3, where they are used to illustrate the present model, are among these. The verbal protocols were propositionalized (according to the standards of Bovair & Kieras, 1985 and following Kintsch, 1974) and used to form the basis for the long-term memory network. They represent the core of what our simulation knows about performing these tasks and can be categorized as two types of information; general knowledge about computers and the tasks to be done, and knowledge about plan elements which are commands to execute in order to accomplish the tasks.

2.1.2 Enrichments to Protocol Contents

This core of long-term memory contents was enriched in three ways. First, six different subjects were asked to provide free associations to the original nodes in the network (i.e. the original propositions

obtained from the protocols), and their responses were added as further nodes to the network. Secondly, for all propositions which stated requests (to enter mail, to send a message, to edit a file, to read a file, etc.) a second proposition was added that stated what the outcome in the world would be if this request were satisfied. This information was needed for the simulation to work, and, although it was sometimes generated spontaneously in the verbal protocols or as a free association, it was necessary to make sure that the system always knew what the results of completing the tasks that it was requested to do would be. Finally, a set of 26 plan elements were included in the long term memory matrix.

Plan elements are the basic actions that the system has available. They are derived from the verbal protocols of users performing the kinds of tasks with which NETWORK deals, such as Edit-a-file, Read-a-File, Reply-to-a-Mail-Message, Enter-Mail, etc. Some additional plan elements were also introduced which were necessary to perform a few tasks for which NETWORK was designed but which subjects were not specifically asked to execute in the protocol study.

A complete list of plan element names is given in Table 1. Planning in our simulation consists of putting these known plan elements together in the right sequence to produce the requested results.

Insert Table 1 about here

Plan elements are formally propositions, like all other nodes in the network. They take three arguments: a plan name (e.g., REPLY to a message), a set of conditions which must exist in the world for this plan element to be executable (e.g., there must be a message to which to reply), and a set of outcomes of the execution of the plan element (e.g., someone receives the reply). The outcome(s) of these plan elements may change dynamically from step to step, depending upon previous actions. For example, the outcome of pasting a text into a buffer will differ depending upon what has happened previously. If the text has already been either pasted or typed into the buffer, the outcome of paste would change from there being a single copy of the text in the buffer to there being two copies of the text in the buffer. Similar outcome changes occur, e.g., to the [EXIT FILE] plan element as editing operations take place.

2.1.3 Connectivity in Long-term Memory

All elements in the simulated long-term memory matrix are connected in a way that is intended to roughly correspond to the associative and semantic relations among the items. The approximation used here is argument overlap: all propositions sharing a common argument are linked, as are propositions which are embedded in other propositions. For all its shortcomings, this procedure has proven reasonably satisfactory in simulations of text comprehension and memory, and is simple and objective computationally. It yields, however, only rough approximations of link strengths. While this is sufficient for our long-term memory simulation, several more specific procedures for estimating link strengths are used in constructing the actual task networks below.

The long-term memory network thus constructed has 81 nodes, yielding a 81 x 81 matrix, the entries of which correspond to the link strength between each pair of nodes. Since this matrix is used only to sample nodes that are related to other nodes, rows are normalized to sum to 1, so that link strengths are directly interpretable as probabilities.

This long-term memory matrix is the general knowledge that NETWORK has of the to-be-performed tasks and the domain related information. When given a text which is a request to perform a certain task, the simulation employs that knowledge to figure out what to do. Specifically, in understanding the task instructions and performing the requested task, NETWORK uses this general knowledge to elaborate and enrich the input it received, and then calculates a solution from the knowledge-enriched textual input.

2.2 Task Networks

2.2.1 Construction of the Task Network

The texts to be understood by NETWORK are always brief requests to perform some routine computing task. Texts were given to the system in propositional form (after Kintsch, 1974), categorized as INTHEWORLD or REQUEST propositions. An example of an INTHEWORLD proposition might be, [EXIST FILE^LETTER INTHEWORLD] to represent that "there is a file called letter in the world". An example of a REQUEST proposition might be [REQUEST [SEND YOU MESSAGE TO-SOMEONE]].

2.2.1.1 Generating the Elements of a Task Network

Given this propositional input, the simulation constructs various knowledge enrichments, using its long-term memory as a source:

(a) Given a REQUEST, the simulation looks in its long-term memory for the corresponding OUTCOME, and adds it to the the proposition list. This is accomplished using a modification of the Raaijmaker & Shiffrin (1981) search of associative memory model. In this model a composite cue is used to search memory, activating only items which are related to all components of the cue. Here, the composite cue consists of two propositions; the REQUEST proposition and a proposition of the form [OUTCOME-OF REQUEST $]. The intersecting search utilizes these propositions to select an appropriate OUTCOME proposition from long-term memory.

(b) For each proposition in a task proposition list, the simulation samples with replacement from long-term memory n related nodes with the probabilities specified by the entries of the long-term memory matrix. These propositions are also added to the proposition list.

(c) All plan elements are bound to relevant task objects in the world, and added to the proposition list. The plan elements in long-term memory have variable slots, such as FILE?, TEXT?, etc. Given an INTHEWORLD proposition such as [EXIST FILE^LETTER INTHEWORLD], the variable FILE? in each plan element would be bound to FILE^LETTER. If there is another FILE in the world, a second plan element would be created, where FILE? is bound accordingly. Thus, depending on the state of the world and the particular task to be done, a varying number of bound plan elements is constructed from the long-term memory set. These three types of knowledge enrichment and the original propositions provide the input for the next process.

2.2.1.2 Connectivity in the Task Representation

Having thus constructed the elements of the task representation, their interrelations are constructed next. There are five specific cases, and a general default rule used to determine these:

(a) Given a text proposition which is a request, it is linked to all plan elements with the same name. Thus, if there is a request to SEND something to someone, this request is linked to all plan elements with the name SEND (there are three such plan elements in Table 1). (On the other hand, if there is a request to INCLUDE something in a file, this request is not linked to any plan element, because there is no plan element with the name INCLUDE.)

(b)
1. The OUTCOME proposition is linked to all plan elements that have the selected outcome. Thus, if the outcome of a request is that someone receives the file "Letter", this proposition is linked positively to all plan elements that have the outcome [RECEIVE SOMEONE FILE^LETTER], i.e. all SEND and REPLY plan elements.
2. Likewise, a plan element which would produce an outcome inconsistent with the requested outcome is given an inhibitory link to the OUTCOME proposition.

(c) All plan elements that have all of their outcomes already existing in the world are inhibited by the corresponding INTHEWORLD proposition(s). For example, the plan element [FIND FILE^LETTER] is inhibited if the proposition [KNOW FILE^LETTER LOCATION INTHEWORLD] exists in the world.

The next two cases concern the interconnections among the plan elements themselves:

(d) If a plan element requires a condition X, this plan has a positive link to all plan elements that produce the outcome X, resulting in a type of causal chaining in the matrix. An example is shown in Figure 1.

(e) If a plan element requires condition X, it has an inhibitory link to all plan elements that have the outcome NOT-X, i.e. would destroy condition X in the world. An example is also shown in Figure 1.

Figure 1 (interconnections among plan elements)

(f) The final case for determining interrelations is a default case, creating a link between all propositions that share a common argument. As discussed above in the section on long-term memory

connectivity, this is used here as a substitute for a more precise specification of the actual associative and semantic relations among the propositions. All propositions sharing a common argument are linked. This includes the case where one proposition is embedded as an argument of another proposition.

The precise link strengths for the six cases described above are parameters to be estimated for the model, subject to certain general constraints. Inhibitory links (Cases b2, c, and e above) must be made relatively strong numerically, because they must not be overwhelmed by the many positive connections in the network. Specific connections (Cases a, b1, and d above) should be stronger than the default connection (Case f). Where no link is specified, connections are set to zero.

It is important to note that these links are created dynamically for each step of a task. Because, as mentioned earlier, previous actions can change plan element outcomes, plan element outcomes can change from producing a specified outcome to not producing that outcome and vice versa. Also, depending upon the actions selected, propositions can either become added to or deleted from the network. The benefits of this dynamic linking are revealed in a subsequent section of this paper.

2.2.2 Integration of the Task Network

Through the above operations a coherence matrix C, consisting of the relations between all pairs of task description, general knowledge and plan element propositions, is created for each task NETWORK is given to do. A vector is created with an element for each proposition, the values of which specify a current activation value for each. Each element of the vector is assigned an initial activation value: the original k text propositions and the proposition representing the selected outcome are assigned a value of 1/k and all other elements (i.e. associated general knowledge and plan elements) are assigned a value of 0. From this starting vector A_1 activation is propagated throughout the coherence network, C, until a final stable activation vector A_Z is obtained. Formally, this means that A_1 is multiplied repeatedly by the matrix C, until it stabilizes, i.e. the average change in activation value is less than some value ε. After each multiplication, elements with negative activation values are set to 0, and all activation values are normalized, so that the sum

of activation remains constant at 1 (Rumelhart & McClelland, 1986). The final pattern of activation among the plan elements given by the resulting activation vector A_Z determines the extent to which NETWORK wants to execute each plan element. A very simple decision rule is employed: execute the most strongly activated plan element in A_Z.

2.2.2.1 Can-do and Want-to Nodes

At this point a complication arises, for frequently the most strongly activated plan element has conditions necessary for its execution which are not satisfied in the world. For instance, one cannot reply to a mail message by sending a file, if there is no file to be sent, or if one is at the system rather than at the mail level. It will not do simply to inhibit all plan elements whose conditions in the world are not satisfied, for understanding the instructions usually requires planning future actions and thinking about hypothetical states of the world. This impasse is solved by creating for each plan element an associated can-do node. Thus, all calculations take place in the network as described above, without regard for what is currently possible in the world and what is not. The most activated plan element that results from these calculations, however, only portrays what the system wants to do. The can-do brothers of these want-to nodes determine what actually happens: they are activated only from their corresponding want-to nodes and are inhibited by in-the-world propositions unless their conditions are satisfied in the world. Thus, one would find in general a state of affairs where the most highly activated want-to planning node is not the same as the most highly activated can-do node. The latter, however, determines what action will be taken immediately, and the former guides what will happen during the course of solution.

There is, however, one further restriction on can-do nodes. If there is a set of plan elements with the same name and outcome (e.g., SEND by typing your message, and SEND by transmitting an existing file both result in someone receiving the text), only the most strongly activated want-to node transmits its activation to its can-do counterpart, while the other "same outcome" can-do nodes are set to a value of 0. Thus, in a field of plan elements that are all alternatives for performing the same action, the system will always insist on the most strongly activated alternative, even though it may not be possible in the present state of the world: NETWORK will try to change the state of the world so as to satisfy the conditions for what

it wants to do, rather than execute a less desirable but feasible alternative.

2.2.2.2 Adding Outcomes to the World: Planning Steps

Suppose that the most highly activated can-do node has been found. Its execution will have a certain outcome. This outcome is represented by one or more propositions (which are specified in the outcome field of each plan element - see Figure 1), which are now added to the network as in-the-world propositions. They are linked to the already existing network by the same rules that were used in creating the original network. Thus, a new expanded network is obtained, with somewhat different interrelationships among the nodes. Most importantly, the newly added in-the-world propositions will now inhibit the plan element that was just executed (by Rule c in Section 2.2.1.2) and any other which would produce the same outcome.

The whole process of integration via spreading activation is now repeated with the new network. As a result, a new can-do plan element will be executed, its outcome will be added to the network as in-the-world propositions, and the integration cycle can be repeated. The process stops when a can-do plan element is executed that produces as its outcome the outcome associated with what was originally requested in the task instructions. That is, of course, if the simulation successfully understands the task. If not, the process will go awry with irrelevant plan elements being executed. At present we are only concerned with a simulation of correct solutions. An error theory is not yet part of our model. The following section explains how NETWORK uses the described processes to understand and plan solutions to a subset of the specific tasks which it has been given.

3. Understanding Tasks

3.1 The INCLUDE Task

3.1.1 The Instructions

NETWORK is presented with the following text: "Include an address that you know in a letter that is in a file, starting at the system level". In propositional form we get

P1 [EXIST FILE^LETTER INTHEWORLD]
P2 [EXIST TEXT^LETTER INTHEWORLD]
P3 [EXIST TEXT^ADDRESS INTHEWORLD]
P4 [IN TEXT^LETTER FILE^LETTER INTHEWORLD]
P5 [KNOW TEXT^ADDRESS CONTENTS INTHEWORLD]
P6 [REQUEST [INCLUDE TEXT^ADDRESS FILE^LETTER] INTHEWORLD]
P7 [AT-LEVEL SYSTEM INTHEWORLD]

Obviously, the step from the text to these seven propositions is a non-trivial one. Several inferences requiring situational knowledge have been made. This aspect of the problem is not dealt with in our model at present, however. NETWORK's attempt at understanding starts with P1-P7. These propositions have been marked as INTHEWORLD, to distinguish them from knowledge elaborations and plan-elements which do not have the same status.

3.1.2 Knowledge Elaborations

There are three components to the knowledge elaboration process. First, the system calculates the outcome of what it has been requested to do. In other words, NETWORK searches long-term memory for the outcome of the request to include a text into a file. It finds [OUTCOME-OF [INCLUDE TEXT^X FILE^Y] [IN TEXT^X FILE^YX]] and binds X to ADDRESS and Y to LETTER. To produce this outcome INTHEWORLD becomes the goal of the system.

Second, NETWORK uses P1-P8 (the original seven task propositions and the newly selected outcome) to sample two associated propositions each from long-term memory[2]. For INCLUDE, only four associates were generated in this way, however, because the sampling with replacement led to frequent duplications, and because there were no long-term memory entries to sample from for certain propositions. None of these associates turned out to have much effect on the further course of events. Thus, associative knowledge elaboration plays only a minor role in the solution of the INCLUDE task. However, this phase plays a major part in understanding other types of discourse (Kintsch, 1988), as well as in the extended NETWORK simulation described in Section 4 below. Third, the whole set of plan elements is selected from long-term memory and used in the subsequent binding process.

---

[2]The number two is arbitrary; in various applications we have tried values between two and ten.

3.1.3 Task-Appropriate Plan Elements

The variables contained in the plan elements available in long-term memory (Table 1) are bound to the appropriate objects INTHEWORLD. Specifically, FILE? is bound to FILE^LETTER, and TEXT? is bound once to TEXT^LETTER and once to TEXT^ADDRESS. Thus, 33 plan elements are created from the original 26 for the task network of the INCLUDE task: all plan elements containing the variable TEXT? are duplicated, once for TEXT^ADDRESS and once for TEXT^LETTER. Then the task connectivity matrix is created using the algorithms previously described in section 2.2.1.2.

3.1.4 Integration Cycles

The relationships between all pairs of the seven INTHEWORLD propositions derived from the text, the outcome proposition, the four associates generated by the knowledge elaboration process, and the 33 can-do and want-to-do pairs of plan elements are used to form the entries of the 78 x 78 matrix **C**. Numerical link strengths were assigned to **C** on the basis of informal trial-and-error explorations for a workable parameter set. Default links via argument overlap were given the least weight, .4. Links among plan elements, which provide the causal chaining, were assigned a value of .7, and request and outcome links were weighted most heavily, 1.5. All inhibitory links were set at -10. This asymmetry was necessary to assure that the few inhibitory connections were not overwhelmed by the many positive links in the network.

A 1 x 78 activation vector $\mathbf{A}_1$ was defined with 1/8 for the first eight INTHEWORLD elements (the seven given text propositions and the selected outcome) and 0 for the remaining elements. $\mathbf{A}_1$ was multiplied by **C** nine times, until the average change in activation values was less than .0001.

Figure 2 shows the final activation values for the plan elements of the network for the INCLUDE task. The original text propositions, which are not shown in the figure, retained a relatively high level of activation, but of more interest is the pattern of activation of the can-do and want-to nodes. NETWORK wants most strongly to PASTE the TEXT^ADDRESS into the FILE^LETTER; its next choice is to TYPE the TEXT^ADDRESS. However, neither of these plan elements have their preconditions satisfied in the world, so that the corresponding can-do

plan elements have activation 0. The strongest can-do plan element is [FIND FILE^LETTER]. It will fire, and, as a consequence a proposition stating that the location of the FILE^LETTER is now known will be added to the INTHEWORLD list.

Insert Figures 2,3,4,5 here

A new integration cycle begins in which the initial activation vector for step 2, A_2, is multiplied by **C** until it stabilizes. The same two want-to plan elements as before are still the most highly activated ones, but they still do not have their preconditions met (Figure 3). However, the third-strongest plan element from the first settled integration cycle, [EDIT FILE^LETTER] now does have its precondition satisfied - we know where the file is - the corresponding can-do is activated and the system therefore enters the editor. This step, once again, changes the state of the world. Not only does it change the level at which NETWORK is operating, but is also results in the creation of several other propositions. These propositions reflect the fact that what was true at the system level of the FILE^LETTER is now true of the BUFFER^LETTER in the editor.

On the third integration cycle NETWORK (Figure 4), although it still cannot act on its first choice because the TEXT^ADDRESS is not in a buffer that could be pasted into FILER^LETTER (a precondition for PASTE), manages to [TYPE TEXT^ADDRESS]. The outcome of this step is that the TEXT^ADDRESS is in a BUFFER^LETTERADDRESS, which is close to what we want (i.e. [IN TEXT^ADDRESS FILE^LETTERADDRESS]), but not quite right yet.

At this time, the outcome of [EXIT FILE] changes. Whereas before the text editing took place,the outcome of [EXIT FILE] was [NOT-AT LEVEL EDIT], this has now been supplemented with an additional proposition which states that executing [EXIT FILE] now will also result in the TEXT^ADDRESS being in the FILE^LETTERADDRESS, the selected outcome for the specified request. This allows it to receive a link from the strongly activated outcome proposition and permits NETWORK to distinguish between quitting and exiting the editor, a difference which it neglected prior to editing (see relative activation values for QUIT and EXIT for steps 1, 2, and 3).

Another integration cycle is therefore performed, with the result of the [EXIT EDITOR] plan element being most active as shown in Figure 5. Exiting the editor has the consequence that we are no longer at

the edit level but at the system level, which is inconsequential for present purposes. It also changes [IN TEXT^ADDRESS BUFFER^LETTERADDRESS] into [IN TEXT^ADDRESS FILE^LETTERADDRESS], which matches the outcome of the original request to include the address in the letter. The task is solved, the instructions have been understood.

3.2 The SEND Task

The second task to be described here does not illustrate any new principles, but is considerably more complex than the INCLUDE task. The to-be-understood instructions are: "You have received a mail message from a friend asking for you to send a paragraph from a file containing a manuscript of yours. Start at the mail level after having read the message." Table 2 shows these instructions in propositional form, the outcome of the request to send the paragraph to someone, and several associative knowledge elaborations, as well as the original set of plan elements that NETWORK created for this task.

In the first step, the can-do node that fires is [ENTER SYSTEM]. There were, however, several want-to plan elements that were more highly activated than the one that could actually be executed. Figure 6 shows the pattern of activation at this point.

Insert Table 2 & Figure 6 & 7 here

At the bottom of Table 2 the outcome of this step is shown, as it has now been added to the list of INTHEWORLD propositions. The next integration step brings [FIND FILE^MANUSCRIPT] to the top of the can-do elements, and once again the state of the world is slightly changed thereby. In Step 3, the system enters the editor, with the result that there is not just a FILE^MANUSCRIPT, but also a BUFFER^MANUSCRIPT. That creates a new set of nodes, as it did in INCLUDE, also shown at the bottom of Table 2, which are added to the network. Next, the TEXT^PARAGRAPH is copied from the FILE^MANUSCRTIPT in Step 4, and now resides in a buffer. In Step 5, the TEXT^PARAGRAPH is pasted into a different buffer, which, in Step 6, becomes the FILE^PARAGRAPH when the system chooses to EXIT the editor (rather than QUIT, which would have had a different outcome). Since there is now a new file INTHEWORLD, all plan

elements dealing with a file are duplicated, and the duplicates are bound to FILE^PARAGRAPH.

Throughout these steps, NETWORK wanted to [SEND TEXT^PARAGRAPH MAIL], as requested. In Step 7, however, this plan element is blocked, because one of its preconditions, [NOT-EXIST FILE^PARAGRAPH INTHEWORLD], has now been violated.[3] On the other hand, the newly created plan element [SEND FILE^PARAGRAPH MAIL] is receiving some activation now. But all NETWORK can do is to [ENTER MAIL]. Finally, in Step 8 [REPLY FILE^PARAGRAPH MAIL] fires (its can-do node is actually tied with the corresponding SEND element, and was chosen randomly, but either one would have had the same effect; REPLY just saves typing in the receivers mail address). The outcome implied by the original request has now been produced, and the process stops. Figure 7 shows the final set of plan element nodes in the network created for the SEND task and their activation values[4].

NETWORK has reasoned its way through eight different steps to arrive at this result. Only the last of these was directly cued by the instructions. The rest were inferred by elaborating the task instructions with what NETWORK knows about the task and the computer system on which it is to be executed, tracking the changes that would occur in the world if certain actions were performed. The processes involved are those ordinarily involved in text comprehension. What is special is that these processes operate within a knowledge-rich domain. A lot of detailed knowledge about what can be done and what the consequences of doing are has made this rather intricate understanding process possible.

3.3 The REVISE Task

---

[3]Previously [KNOW TEXT^PARAGRAPH CONTENTS INTHEWORLD] was missing. This was produced by the COPY plan element but before REPLY can take advantage of it, this other violation occurs.

[4]Here the addition of the semantic information mentioned in section 2.2.1.2 makes a difference in NETWORK's solution. Because it was selected as general information during knowledge activation, the proposition [ISA REPLY-COMMAND SEND-COMMAND] allows the request to become bound to REPLY as well as SEND. Activity thus flows from the REQUEST to the REPLY plan element, which in turn further activates [ENTER MAIL], one of its preconditions. Without this knowledge, NETWORK would try to SEND the file at the system level instead.

In Kintsch & Mannes (1987) it has been observed that subjects providing verbal protocols of scriptal actions, for example "buying groceries", employed special linguistic signals such as "and then" to mark the major boundaries between the scriptal units. We have therefore assumed that if such a linguistic marker occurs in an instruction, it would lead the comprehender to segment the task at this point into separate pieces. Thus, the instruction "You are to revise a manuscript you are working on with a colleague by removing a paragraph, and then send the revised manuscript to that person" should be interpreted as two separate tasks, to be performed in sequence.

That is how NETWORK approaches this task. It reads only up to the "and then", and proceeds to revise the manuscript as requested: it first finds the FILE^MANUSCRIPT, enters the editor, cuts the TEXT^PARAGRAPH from the manuscript, and creates a file containing the revised manuscript when exiting the editor. This finishes the first part of the task, and NETWORK now reads the remainder of the instructions. In the final integration cycle [SEND FILE^MANUSCRIPT SYSTEM] turns out to be the strongest want-to as well as can-do node, and the problem is solved[5].

If the "and then" in the task instructions is neglected, NETWORK still understands the REVISE task, with the main difference being that the SEND-element is now strongly activated from the very beginning. However, longer, more complex tasks very likely could not be solved without explicit linguistic markers that show how the problem can be segmented.

3.4 The PRINT-AND-DELETE Task

Conflict resolution has long been of interest in the planning literature. A well known example of a problem with conflicting subgoals is "paint the ceiling and paint the ladder" (Sacerdoti, 1977). "Print and delete the file eggplant" is an analogue to this problem in NETWORK's domain. This class of problems has often proved difficult for artificial intelligence programs to handle because of the explicit task subgoaling procedures these programs use. Standard subgoaling procedures would separate the compound task into its constituent

---

[5]Note that in solving this task NETWORK chose to send a file from the system level. In the SEND task, there was a mail message to reply to so NETWORK had the opportunity to reenter the mail system in order to complete the request.

subparts, painting the ceiling and painting the ladder, with no immediate regard for interactions among these parts. Sophisticated conflict resolution procedures must then be employed, as in the work of Sussman (1975), to take task sub-part interdependencies into account.

Figure 8 shows how NETWORK handles this situation. For brevity, only a subset of relevant plan elements are shown. In the first step, PRINT is most highly activated among the want-to nodes, but not DELETE, because it is inhibited by PRINT (deleting the file would remove a precondition for printing it just as it does for finding it, see Figure 1). Thus, NETWORK first finds the file, prints it in the second step, and, with the inhibition from PRINT removed because [PRINT FILE^EGGPLANT] itself is now inhibited by the already existing printed copy of that file INTHEWORLD, deletes it in the third step. Here the causal chaining has done the work which typically required resolution procedures.

Insert Figure 8 here

---

4. Case-Based Reasoning

The plan elements are generalized, decontextualized knowledge expert computer users are presumed to have and employ in understanding tasks like the ones described above. It is, however, entirely unreasonable to assume that experts who have learned these rules do not also have any episodic memory traces of having solved such problems. It is equally implausible that such memory traces should play no role in their understanding of similar problems. Indeed, the large literature on reminding provides many illustrations of the importance that the memory of specific problem solving episodes plays (e.g. Schank, 1982). Current conceptualizations of case-based reasoning are strongly influenced by the seminal work of Kolodner in this field (e.g. Kolodner & Simpson, 1984). Basically, case-based reasoning systems attempt to retrieve plans from memory, select a relevant one, modify it as needed, and evaluate its usefulness. There is no doubt that this kind of reasoning sometimes occurs in people's problem solving, and that it can be an effective artificial intelligence device. There are also, however, other ways in which episodic memory can influence the understanding process, less

deliberate ones, which do not make extensive demands on processing resources.

In a comprehension model like the present one, remembered episodes become themselves part of the associative network, thereby influencing the course of future events. Remembered problem solving episodes will be sampled during the construction of a task representation through the process of associative knowledge elaboration. In fact, the more similar a new task is to a remembered episode, the more likely it will be that the remembered episode will be retrieved from long-term memory, because the representation of the memory trace and the propositional representation of the to-be-understood text share a considerable amount of argument overlap. Once incorporated in the task representation, the memory episode becomes linked with plan elements that were used in past solutions of the task, often changing the spread of activation, sometimes considerably. The effects might be beneficial, if the remembered plan elements are, in fact, relevant in the new situation, or they might misdirect the whole process if they are not.

This is a quite different model than most other approaches to case-based reasoning. These often require selection, evaluation, and modification processes which can be difficult to implement. In NETWORK, there are no case selection processes, no case evaluation, and no modification of plans by analogy, etc. Because cases are selected from long-term memory on purely associative grounds, no new selection procedure must be introduced for cases and because a number of cases can play a role in selectively activating certain plan elements, no case evaluation procedures are required to choose among alternatives. Finally, NETWORK does not choose a case and follow it in completing a task. Hence there are no plan modification techniques because we don't run into the situation of there being no appropriate case to follow. When NETWORK understands a problem text, we simply add a trace of that episode to its long-term memory, thus changing the context for future understanding. We need no new model of case-based reasoning at all, rule-based and case-based reasoning involve the same processes in NETWORK.

There is a problem, however, in deciding exactly what information is contained in a memory episode. It is clear that it cannot be the whole processing trace of the solution process. However, it is not clear exactly which aspects of the process are represented in the memory trace. The available psychological data are not sufficiently

informative on this point. A perusal of the AI and case-based reasoning literature did not reveal any obvious principles for case construction either. We therefore were reduced to making some quite arbitrary choices. We decided to construct memory episodes which consisted of the arguments of three propositions from a task network which were not plan elements and which were most highly activated at the time the task was completed, plus the plan elements that were fired in the course of planning the task. For example, the memory trace formed as a result of solving the INCLUDE problem (Section 3.1) consists of the episode name, the six arguments of the three most strongly activated non-plan element propositions, plus the four plan elements (in italics) needed for that task:

[CASEINCLUDE TEXT^LETTER FILE^LETTER TEXT^ADDRESS FILE^LETTER TEXT^ADDRESS FILE^LETTER *FIND EDIT TYPE EXIT*]

Note that FILE^LETTER appears three times as an argument of CASEINCLUDE, and hence will be linked with any proposition or plan element containing FILE^LETTER three times, with a resulting link strength of 1.2 instead of .4 as would result from a single argument overlap. Therefore, any new problem involving a FILE^LETTER will tend to make NETWORK want to FIND, EDIT, TYPE and EXIT again. (Note that the sequence in which these arguments occur in the case propositions is irrelevant and just happens to follow the temporal order of the solution in this example.)

Insert Figure 9 here

Figure 9 shows the performance of NETWORK on the first step of the REVISE task (Section 3.3) with and without memory for cases. The only memory for cases NETWORK had at this point were for the INCLUDE, SEND, and REVISE (two part) tasks. Of these four cases, three were actually picked up by the knowledge elaboration process (Section 3.1.2): the two REVISE episodes and the SEND episode. Thus, the system remembered having done this problem before, but it also remembered having done a similar problem that also involved sending a mail message and dealt with a manuscript. Because these remembered cases are linked strongly to both the text propositions and plan elements, they become strongly activated. Appropriately, the REVISE1 episode is the most strongly activated element in the task representation at this point. However, as far as the want-to

plan elements go, their overall pattern of activation is similar to the non-case based pattern. The presence of cases served mostly to further increase the strength of relevant plan elements, like FIND and decrease that of irrelevant ones (like READ and FIND MESSAGE). On the other hand, the irrelevant plan element [PASTE TEXT^PARAGRAPH BUFFER] which was quite weak in the original REVISE solution is also strengthened because it gets activation from the SEND episode, where it was relevant. Thus, by allowing cases to support different plan elements to different degrees, neither case selection nor evaluation procedures are necessary. This is desirable because it requires no extension of the model to incorporate cases and make them useable. We presume, however, that given a richer episodic memory, fatal interference effects could arise in this way. Of course, since we are concerned with a simulation of how real people solve these routine problems, similar interference effects can probably be observed in people, too.

The most attractive feature of the treatment of case-based reasoning within NETWORK is the fact that we did not need to introduce any new machinery to deal with it. NETWORK doesn't really make a distinction between rule-based and case-based reasoning. Nor does it distinguish between generic, semantic knowledge (e.g. we write letters to friends) and episodic or case knowledge (e.g. I sent a letter to Mike yesterday) in contrast to what has been suggested by Tulving (1972). Remembered cases are simply part of the context in which instructions are understood. They help or interfere with understanding in just the same way as, say, textual information would.

At present it is not possible to provide a fuller treatment of case memory within NETWORK. The research findings needed to decide many of the open questions are unfortunately not yet available. How much and what of an episode is actually remembered? What is the course of forgetting? To what extent do generalizations occur? (At present, having printed and deleted the FILE^EGGPLANT would be of no use when told to do the same for the FILE^LETTER.)

5. Conclusion

NETWORK does some interesting things, in interesting ways. It is not meant to be an AI program to help us solve routine computing tasks - although some of the ideas developed here might turn out to be useful in that respect. NETWORK is a simulation of how people

understand and perform such tasks. It is, however, only loosely rooted in empirical fact: its basic notions are derived from human protocols, but no attempt has as yet been made to validate the model empirically. It is, rather, an elaborate Gedankenexperiment which includes a computer simulation. On the one hand, such a project can be seen as a necessary preparatory step for further empirical and theoretical research on understanding instructions as texts and for testing the model, as will be suggested below. Equally important, however, is another feature of the research reported here. By exploring in detail how a theory of discourse comprehension can account for planning behavior, we have taken a significant step towards delimiting the proper, somewhat expanded, domain of comprehension theories.

The performance of routine tasks such as the ones studied here cannot be understood in terms of fixed scripts. Any model for these tasks must be generative, like models of language production, and for much the same reason. Familiar elements are recombined in ever novel ways. We have seen above that NETWORK can do that; Mannes (1989) has shown that NETWORK has no trouble performing a truly unique nonsense task, planning a sequence of actions that would make no sense to a user. But just as an experienced user could do so upon request, NETWORK was able to do so also, while it would be absurd to assume the existence of a script for a nonsensical action sequence. Fixed scripts cannot be the solution for these problems, and, as we have argued elsewhere (Kintsch & Mannes, 1987), they do not provide a satisfactory solution in other situations which require scriptlike knowledge either. Structures like scripts must be generated themselves.

One type of process that can generate such structures has been described by problem solving theories. For specificity, we take these to be the theories directly or indirectly derived from the General Problem Solver of Ernst and Newell (1969), although they have, of course, a longer history than that. Generally, these theories describe processes that are under conscious control at least in part, that are relatively resource demanding, that are often time consuming, and that are characteristic of tasks which are typically experienced as difficult, e.g. the cannibals and missionaries problem. On the other hand, the processes involved in performing routine computing tasks are more or less automatic, non-demanding, rapid, and easy. In terms of their subjective experience, they are more like comprehension or perception than "real" problem solving. We have

shown here that, in accordance with that subjective impression, these processes can, indeed, be modelled by a theory of comprehension as a largely automatic process. This in no way proves that the comprehension view is more adequate than the problem solving interpretation, but we at least provide a specific, worked-out alternative to the traditional view.

The construction-integration model of discourse comprehension emphasizes bottom-up processes. It does not try to understand a text by attempting to capture it with an already derived interpretation from a ready-made schema over it. Instead, it carefully analyzes the input it receives and generates a structure appropriate for it. NETWORK chooses plan elements in response to the dynamically changing state of the world, making planning more like a reaction to the environment than a conscious process which is divorced from the world. NETWORK lets the environment help to guide the planning. The situation is not a static constraint to NETWORK, but it is the continually changing context which drives the action. This sort of interleaving of planning and action has been advocated for planning systems before, e.g. by McDermott (1978) and Agre & Chapman (1987). Indeed, NETWORK probably comes closer to a model of situated cognition (Greeno, 1989) than traditional accounts of problem solving.

NETWORK reacts to changes in the situation, somewhat like persons who have acquired some routine with the Tower of Hanoi puzzles do when they let their moves be cued by the actual configuration of the stacks (Simon, 1975), or in the examples of display-based problem solving-such as making coffee-discussed by Larkin (1989). There is, however a difference that should be of considerable interest to system designers between environmental events such as moving a disk to another peg, or putting a filter into the filter holder, and finding a file, or exiting the editor. The former are perceptually salient, so that changes in the environmental state can hardly be missed and thus can safely guide behavior. The events NETWORK is concerned with, on the other hand, are signalled either by subtle perceptual cues, or have no direct perceptual effects at all (the contents of a buffer are now in a file) the user has to know what happens! NETWORK, therefore, models the experienced user; to allow a novice to use a computer system more like an experienced user one would have to redesign the system so as to shift much of the burden of keeping track of the changing environment from the user's knowledge to the external world.

Future work on NETWORK will concentrate most urgently on providing an error theory for routine computing tasks. Only at that point would a detailed empirical evaluation of the model be feasible, along the lines of what Cummins et al. (1988) did for word arithmetic problems. The consequences of specific bugs (e.g. what if there is no causal facilitation among plan elements, what if inhibitory links are too weak, etc.?) on NETWORK's performance could be explored and compared with human error data.

Psychological research on memory for cases might put the use of case-based reasoning in NETWORK on a more stable foundation, as discussed in Section 4. In addition, to make NETWORK even more reactive to its environment, a richer representation of that environment and its non-propositional aspects will be required. Finally, there are questions as to how learning might be incorporated in a system like NETWORK, which we don't understand at present, or how explanations could be incorporated into NETWORK, which seems rather more obvious, in principle.

Thus, the NETWORK project has rich developmental possibilities. Of course, NETWORK does not stand alone, but is part of an effort to model human understanding in a variety of situations in terms of the construction-integration model. The theory of comprehension should have the same central role within cognitive science as understanding texts and situations has human cognition.

References

Agre, P. E. & Chapman, D. (1987). What are plans for? Paper prepared for the Panel on Representing Plans and Goals, DARPA Planning Workshop, Santa Cruz, CA.

Bovair, S. & Kieras, D. E. (1985). A guide to propositional analysis for research on technical prose. In B. K. Britton and J. B. Black (Eds.), Understanding expository text. Hillsdale, NJ: Erlbaum.

Collins, A. M. & Loftus, E. F. (1975). A spreading-activation theory of semantic processing. Psychological Review, 82, 407-428.

Cummins, D., Kintsch, W., Reusser, K., & Weimer, R. (1989). The role of understanding in solving word problems. Cognitive Psychology, 20, 405-438.

Ernst, G. W. & Newell, A. (1969). GPS: A case study in generality and problem solving. New York: Academic Press.

Goetz, E. T., Anderson, R. C., & Schallert, D. L. (1981). The representation of sentences in memory. Journal of Verbal Learning and Verbal Behavior, 20, 369-385.

Greeno, J. G. (1989). Situations, mental models, and generative knowledge. In D. Klahr and K. Kotovsky (Eds.), Complex information processing. Hillsdale, NJ: Erlbaum. Pp. 285-319.

Kintsch, W. (1974). The representation of meaning in memory. Hillsdale, NJ: Lawrence Erlbaum.

Kintsch, W. (1988). The role of knowledge in discourse comprehension: A construction-integration model. Psychological Review, 95, 163-182.

Kintsch, W. & Greeno, J. G. (1985). Understanding and solving word arithmetic problems. Psychological Review, 92, 109-129.

Kintsch, W. & Keenan, J. M. (1973). Reading rate and retention as a function of the number of propositions in the base structure of sentences. Cognitive Psychology, 5, 257-274.

Kintsch, W. & Mannes, S. (1987). Generating scripts from memory. In J. Hoffman and E. van der Meer (Eds.), Knowledge aided information processing. Amsterdam: North-Holland Publishers.

Kolodner, J. L. & Simpson, R. L. (1984). Experience and problem solving: a framework. In Proceedings of the Sixth Annual Conference of the Cognitive Science Society.

Larkin, J. H. (1989). Display-based problem solving. In D. Klahr and K. Kotovsky (Eds.), Complex information processing. Hillsdale, NJ: Erlbaum. Pp. 319-341.

Mannes, S. M. (1989) Problem-solving as text comprehension: A unitary approach. Doctoral Dissertation, University of Colorado.

McDermott, D. (1978). Planning and acting. Cognitive Science, 2, 71-109.

Minsky, M. (1975). A framework for representing knowledge. In P. H. Winston (Ed.), Psychology of computer vision. New York: McGraw-Hill.

Raaijmaker, J. G. & Shiffrin, R. M. (1981). Search of associative memory. Psychological Review, 88, 93-134.

Rumelhart, D. E. & McClelland, J. L. (1986). Parallel distributed processing: Explorations in the microstructures of cognition. Volume 1. Cambridge, MA: MIT Press.

Rumelhart, D. E., Smolensky, P., McClelland, J. L., & Hinton, G. E. (1986). Schemata and sequential thought processes in PDP models. In J. L. McClelland and D. E. Rumelhart (Eds.), Parallel distributed processing: Explorations in the microstructures of cognition. Volume 2. Cambridge, MA: MIT Press.

Sacerdoti, E. D. (1977). A structure for plans and behavior. New York: Elsevier North-Holland.

Schank, R. C. (1982). Dynamic memory. Cambridge: Cambridge University Press.

Schank, R. C. & Abelson, R. P. (1977). Scripts, plans, goals, and understanding. Hillsdale, NJ: Erlbaum.

Simon, H. A. (1975). The functional equivalence of problem solving skills. Cognitive Psychology, 7, 266-268.

Stefik, M. (1981). Planning with constraints (MOLGEN: Part 1). Artificial Intelligence, 16, 111-140.

Sussman, G. J. (1975). A computer model of skill acquisition. New York: Elsevier North-Holland.

Tulving, E. (1972). Episodic and semantic memory. In E. T. Tulving and W. Donaldson (Eds.), Organization of memory. New York: Academic Press.

Three plans in their generic (unbound) form.

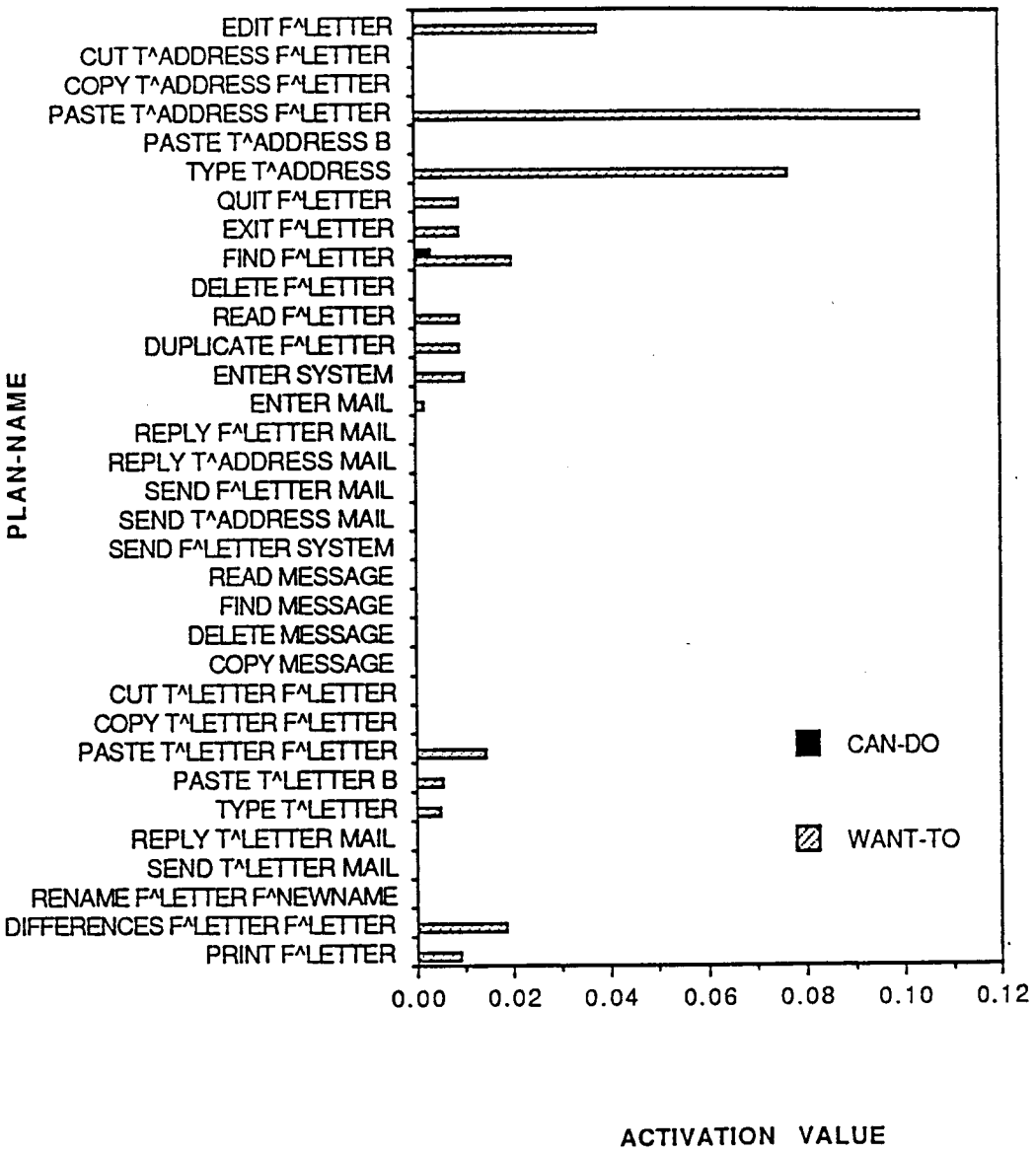
NAME	(FIND FILE)
PRECONDITION(S)	(AT-LEVEL SYSTEM) (EXIST FILE)
OUTCOME(S)	(KNOW FILE LOCATION)

NAME	(DELETE FILE)
PRECONDITION(S)	(AT-LEVEL SYSTEM) (EXIST FILE)
OUTCOME(S)	(KNOW FILE LOCATION)

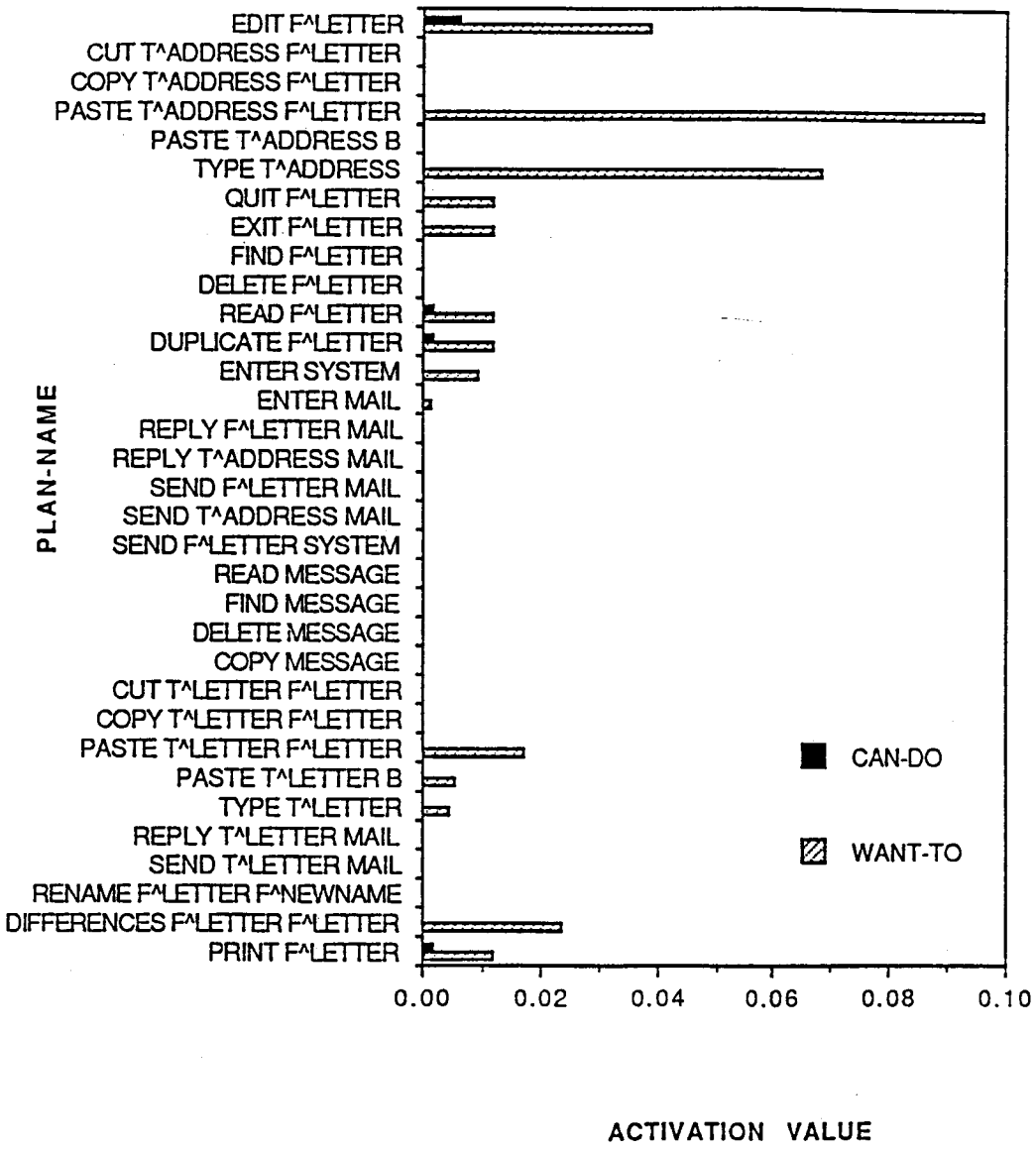
NAME	(PRINT FILE)
PRECONDITION(S)	(AT-LEVEL SYSTEM) (EXIST FILE) (KNOW FILE LOCATION)
OUTCOME(S)	(EXIST HARDCOPY FILE)

FROM THE PLAN ELEMENT REQUIRING A SPECIFIC OUTCOME	TO A PLAN ELEMENT PROVIDING THAT SPECIFIC OUTCOME		
	FIND	DELETE	PRINT
FIND		-	
DELETE	+		
PRINT	+	-	

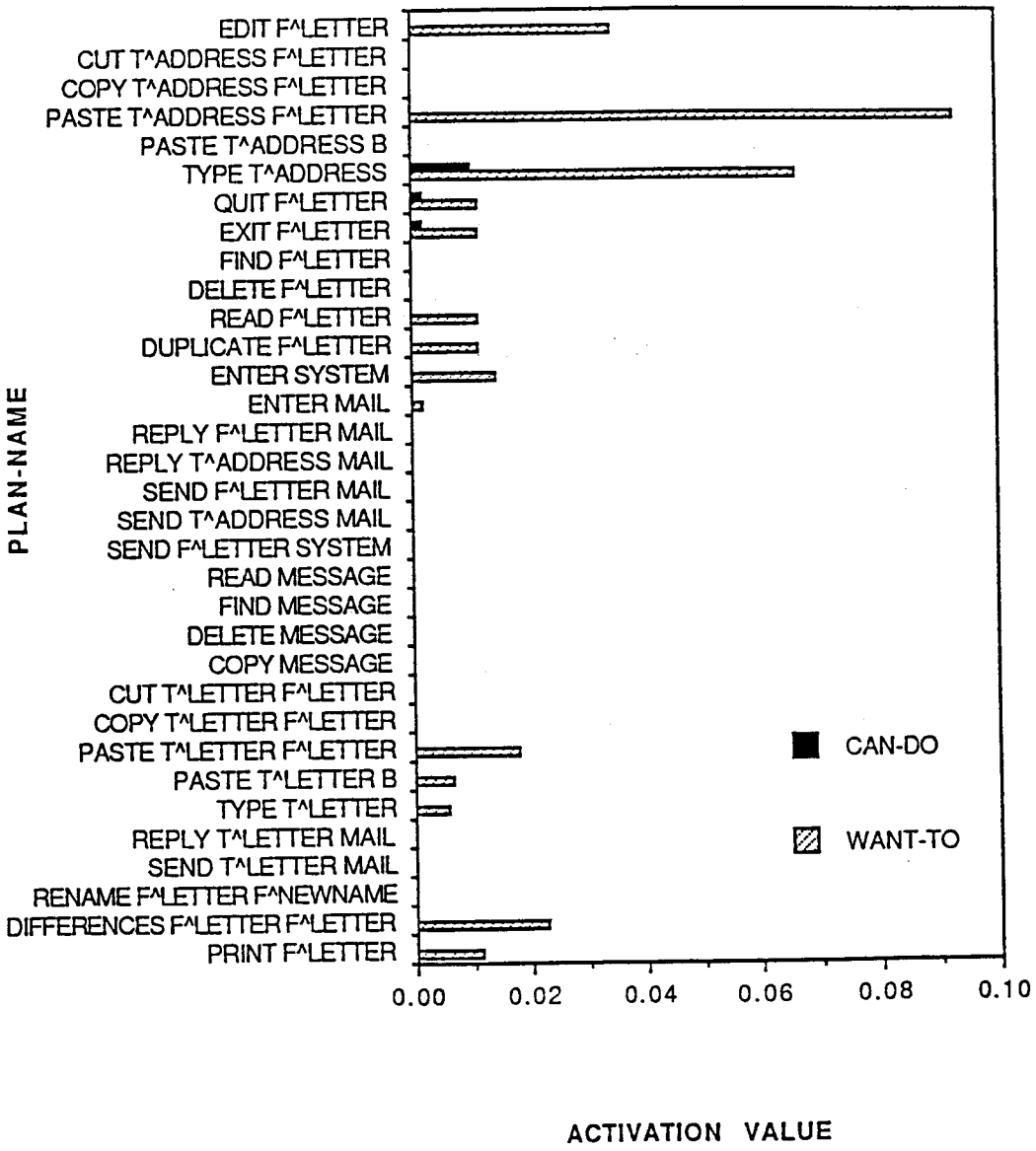
1. A sample of three plan elements are shown in the top panel and the resulting causal chain in matrix form in the bottom panel.



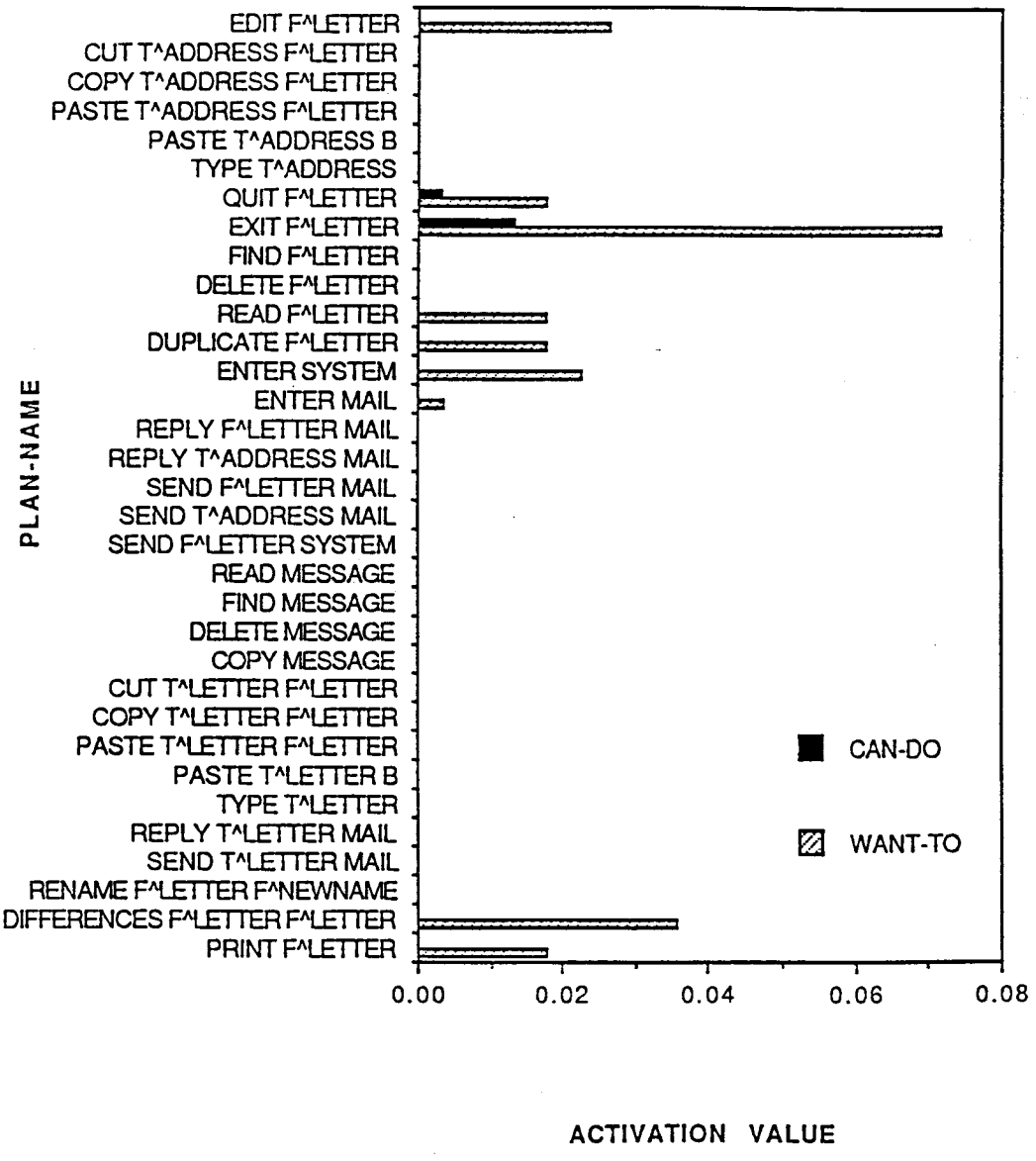
2. Activation values for all plan elements for the first step of the INCLUDE task. Decisions are based on the can-do values.



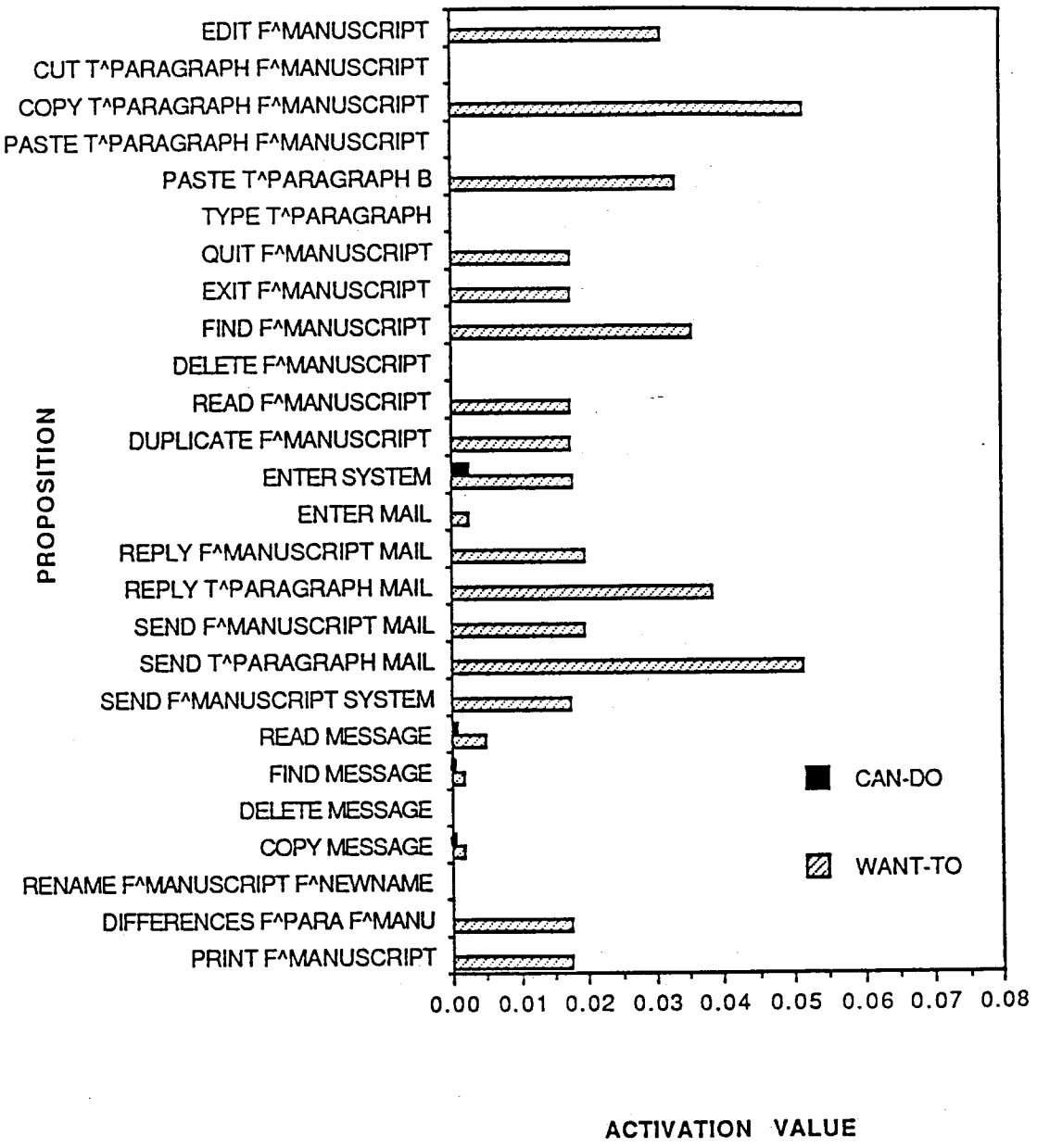
3. Activation values for all plan elements for the second step of the INCLUDE task. Decisions are based on the can-do values.



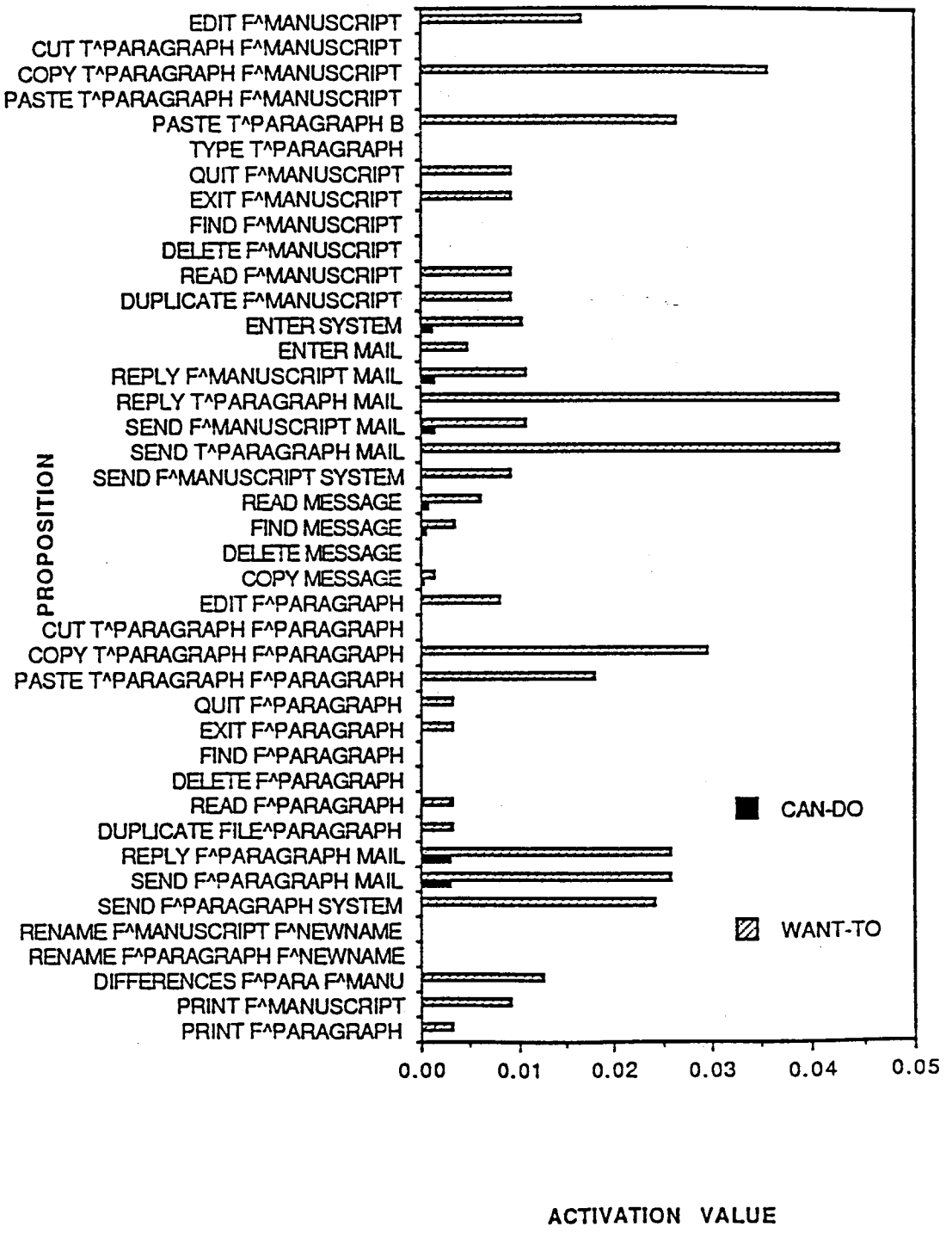
4. Activation values for all plan elements for the third step of the INCLUDE task. Decisions are based on the can-do values.



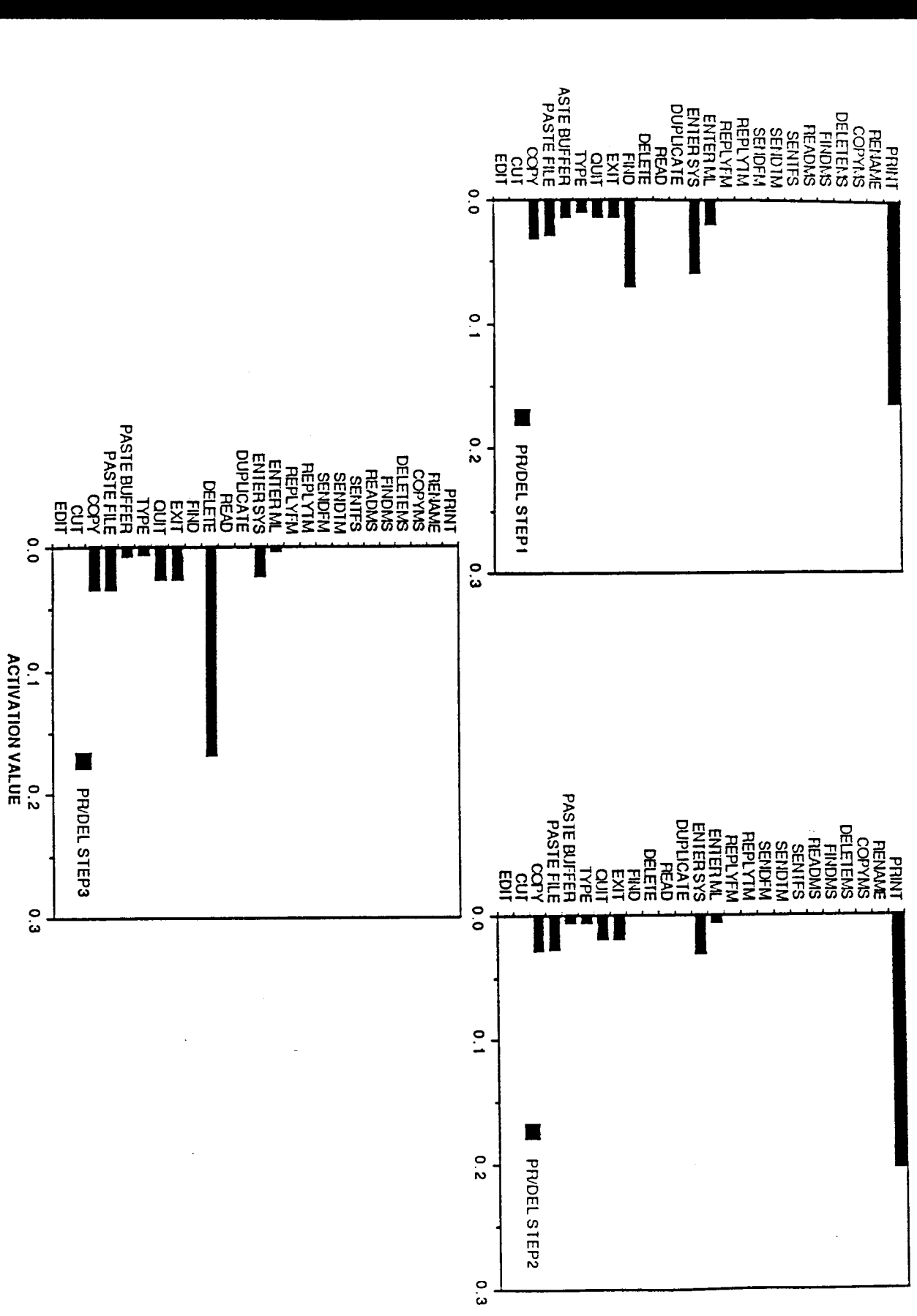
5. Activation values for all plan elements for the fourth step of the INCLUDE task. Decisions are based on the can-do values.



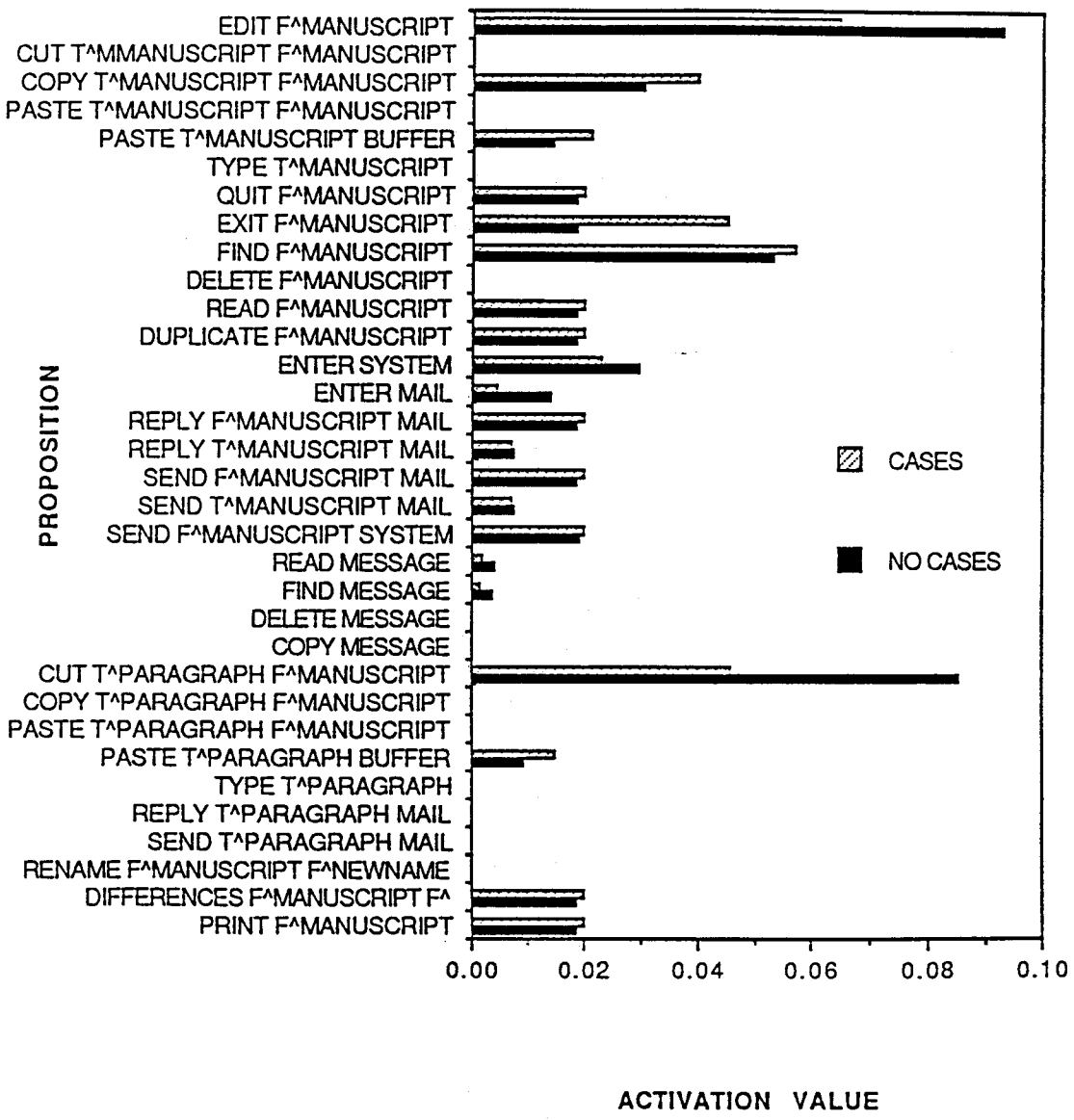
6. Activation values for all plan elements for the first step of the SEND task. Decisions are based on the can-do values.



7. Activation values for all plan elements for the last step of the SEND task. Decisions are based on the can-do values.



8. Activation values for all plan elements for the three steps of the PRINT/DELETE task.



9. Activation values for the first step of REVISE using NETWORK with and without cases.

Table 1

Plan element names

(EDIT FILE?)
(CUT TEXT? FILE?)
(COPY TEXT? FILE?)
(PASTE TEXT? FILE?)
(PASTE TEXT? BUFFER)
(TYPE TEXT?)
(QUIT FILE?)
(EXIT FILE?)
(FIND FILE?)
(DELETE FILE?)
(READ FILE?)
(DUPLICATE FILE?)
(ENTER SYSTEM)
(ENTER MAIL)
(REPLY FILE? MAIL)
(REPLY TEXT? MAIL)
(SEND FILE? MAIL)
(SEND TEXT? MAIL)
(SEND FILE? SYSTEM)
(READ MESSAGE)
(FIND MESSAGE)
(DELETE MESSAGE)
(COPY MESSAGE)
(RENAME FILE? FILE^NEWNAME)
(DIFFERENCES FILE? FILE?)
(PRINT FILE?)

Table 2

Contents of the Input List for SEND

Task Description
(P13 (EXIST TEXT^MANUSCRIPT INTHEWORLD))
(P14 (EXIST TEXT^PARAGRAPH INTHEWORLD))
(P15 (EXIST FILE^MANUSCRIPT INTHEWORLD))
(P16 (EXIST MESSAGE INTHEWORLD))
(P17 (KNOW MESSAGE LOCATION INTHEWORLD))
(P53 (IN MESSAGE MAIL INTHEWORLD))
(P18 (READ MESSAGE INTHEWORLD))
(P19 (IN TEXT^PARAGRAPH FILE^MANUSCRIPT INTHEWORLD))

Task Request
(P20 (REQUEST MESSAGE P21 INTHEWORLD))
(P21 (RQSEND TEXT^PARAGRAPH INTHEWORLD))

Task Outcome Selected from Long-term Memory
(P22 (OUTC21RECEIVE TEXT^PARAGRAPH))

Some General Knowledge
(P36 (USE MAIL STUDENTS))
(P52 (NOTIN TEXT^PARAGRAPH FILE^MANUSCRIPT))
(P34 (IS MESSAGE SHORT))
(P26 (OUTC25NOTIN TEXT^PARAGRAPH FILE^MANUSCRIPT))
(P49 (KNOW FILE^MANUSCRIPT LOCATION))
(P56 (ISA REPLY-COMMAND SEND-COMMAND))
(P51 (IN TEXT^PARAGRAPH BUFFER))
(P31 (SUBMIT MANUSCRIPT))
(P30 (ISA TEXT^PARAGRAPH TEXT))
(P43 (IS TEXT^PARAGRAPH LONG $P22))

The Original Set of Plan-elements
(L1 (EDITF FILE^MANUSCRIPT))
(L2 (CUTTF TEXT^PARAGRAPH FILE^MANUSCRIPT))
(L3 (COPYTF TEXT^PARAGRAPH FILE^MANUSCRIPT))
(L4 (PASTETF TEXT^PARAGRAPH FILE^MANUSCRIPT))
(L5 (PASTETB TEXT^PARAGRAPH BUFFER))
(L6 (TYPET TEXT^PARAGRAPH))
(L7 (QUITF FILE^MANUSCRIPT))
(L8 (EXITF FILE^MANUSCRIPT))
(L9 (FINDF FILE^MANUSCRIPT))

(L10 (DELETEF FILE^MANUSCRIPT))
(L11 (READF FILE^MANUSCRIPT))
(L12 (DUPLICATEF FILE^MANUSCRIPT))
(L13 (ENTERSYS SYSTEM))
(L14 (ENTERML MAIL))
(L15 (REPLYFM FILE^MANUSCRIPT MAIL))
(L16 (REPLYTM TEXT^PARAGRAPH MAIL))
(L17 (SENDFM FILE^MANUSCRIPT MAIL))
(L18 (SENDTM TEXT^PARAGRAPH MAIL))
(L19 (SENDFS FILE^MANUSCRIPT SYSTEM))
(L20 (READM MESSAGE))
(L21 (FINDM MESSAGE))
(L22 (DELETEM MESSAGE))
(L23 (COPYM MESSAGE))
(L24 (RENAME FILE^MANUSCRIPT FILE^NEWNAME))
(L25 (DIFFERENCES FILE^MANUSCRIPT FILE^))
(L26 (PRINT FILE^MANUSCRIPT))

Outcome Propositions from Steps One through Three
(P41 (AT-LEVEL SYSTEM INTHEWORLD)) Step 1
(P201(KNOW FILE^MANUSCRIPT LOCATION INTHEWORLD)) Step 2
(P202(AT-LEVEL EDIT BUFFER^MANUSCRIPT INTHEWORLD)) Step 3
(P203(EXIST BUFFER^MANUSCRIPT INTHEWORLD)) Step 3
(P204(IN TEXT^PARAGRAPH BUFFER^MANUSCRIPT INTHEWORLD)) Step 3