Routine Computing Tasks: Planning as Understanding

SUZANNE M. MANNES AND WALTER KINTSCH

University of Colorado

A system called NETWORK is described which implements the construction-integration model of Kintsch (1988) in a routine computing-task domain. This system builds a plan of action on-line for a given task from a set of plan elements. These plan elements are simple overlearned production rules that are put together by NETWORK to produce plans for novel tasks. This approach is contrasted with other types of planners as NETWORK is shown to plan solutions to a variety of tasks. Discussions focusing on the use of long-term memory, case-based reasoning, and planning and acting are presented.

NETWORK takes as input a task description, uses this information to select related knowledge from its long-term memory, and constructs a network representation of the task. This network is then integrated through a spreading-activation procedure where irrelevant items in the network become deactivated, and things that appear related sustain each other's higher activation. Subsequently, a decision process chooses a plan element for firing, depending upon its level of activation with those more highly activated being considered for action first. When a plan element is found that can fire, its outcomes are added to the state of the world. The process repeats until a selection of plan elements is produced to complete the task.

1. INTRODUCTION

A recent development in the field of cognitive science is the theoretical framework put forth by Kintsch (1988) on the construction-integration model. This text-comprehension model provides a convenient and parsimonious way of looking at human performance on a variety of cognitive tasks. Because the theory has been derived from text-comprehension data, it is not surprising that many of the tasks for which it is relevant are precisely

This research was supported by the Army Research Institute, ARI Project MDA903-86-CO143. The work reported here was part of the Ph.D. 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. We would also like to thank Michele Roushey for her technical expertise as well as Richard Alterman and three anonymous reviewers for insightful comments on an earlier version of this article.

Correspondence and requests for reprints should be sent to Suzanne Mannes, Department of Psychology, University of Delaware, Newark, DE 19716.
those on which its development depended. Data from students verifying sentences of various types (Kintsch, Welsch, Schmalhofer, & Zimny, 1990), story understanding (Kintsch, in press), and priming data (Kintsch, 1988; Kintsch & Welsch, in press) are among these. In addition, however, this model, and the framework it provides for understanding human information processing, can be applied to domains which, on the surface, seem not to be matters of text comprehension at all. One of these domains, the solution of routine computing tasks, is studied here. We present a simulation, NETWORK\(^1\), which embodies the construction-integration model and produces sequences of action selections that form plans to solve computing tasks. Finding solutions for these tasks has, in the past, been relegated almost exclusively to planning systems, not understood as instances of text comprehension. The remainder of this section contains a brief review of some of the relevant planning literature. The next section is a description of our model. This is followed by a report of NETWORK’s performance on several tasks, and a discussion of how it relates to other planners.

Planners can be categorized along a number of dimensions. They can be divided into those that create a plan anew and those that reuse and modify old plans for each task, those that deal with real-world problems, problems someone might encounter on a regular day, and those that deal with “toy” problems, problems or domains created especially for the purpose of investigating planning behavior. There are planners that attempt problems which are solved almost automatically by humans, and some that attempt problems which require conscious effort; those which utilize knowledge of the planning process itself and those which depend solely upon knowledge of their problem domain; and those which exhibit hierarchical representations of planning and those which do not.

Presently, NETWORK constructs plans on-line for each task anew. Thus, a discussion of case based reasoners is left to the discussion section where the case-based possibilities for NETWORK are addressed. Of those planners which create plans or solve problems anew each time, perhaps the best known is GPS (Ernst & Newell, 1969). In GPS, problem solving is conceived of as the solver navigating around in a search space via a set of operators. Each operator selection is guided by means-end analysis that reduces, via a table of connections, the difference between the current and goal states. The tasks and the domain-specific knowledge used to solve a problem are separate from the general problem-solving procedures GPS possesses. Hence, the variety of tasks GPS can handle is quite large (11 different classes), and the methods quite general. The problems with which it deals “seem

\(^1\) Note that NETWORK will be used here to refer to the computer simulation, and network will be used to refer to semantic networks as they are traditionally understood.
quite formal-mathematical problems and puzzles” (Ernst & Newell, 1969, p. 268). This class includes both real-world problems and toy problems, such as proving theorems in predicate calculus and water jug problems, respectively: Solutions all require conscious effort by a human solver.

In STRIPS (Fikes & Nilsson, 1971), the means–end analysis framework for problem solving has been used to produce the more common, everyday behaviors that might be required of a robot navigating and arranging objects in a complex environment. STRIPS uses a set of basic operators to build a solution and uses a means–end strategy to search through a space of possible world models. Whenever an operator is executed, its effects are represented in two lists, DELETIONS and ADDITIONS.

Because the tasks with which planners deal are often complex, many attempt to reach a solution via subgoal decomposition making it practical to accept the linearity assumption (Sussman, 1975). That is, to assume that none of the subgoals, which have just been developed, interact in undesirable ways. A view of the problem must be taken so that it appears as a nearly decomposable system (Simon, 1969). To the extent that any task or task domain is not decomposable, and subgoals do interact, backtracking may be required. ADDITION and DELETE lists make it possible to get back to a previous or even the original problem state via backtracking when an erroneous move has been made, but backtracking is a costly process, wasting both time and resources. It is characteristic for nonhierarchical systems—planning at all levels of detail at once in an essentially depth-first manner—to suffer from complications arising from interacting subgoals.

The problem of conflicting subgoals has been addressed in a number of ways. In particular, HACKER (Sussman, 1975) maintains a set of problem-solving routines, critics, to identify and correct problems of conflicting, interacting, or conjunctive subgoals. HACKER’s planning strategy encourages developing a plan that is perhaps buggy, due to the acceptance of the linearity assumption, and then fixing it, instead of trying to produce a perfect plan initially. HACKER can thus use old plans from its answer library or construct new ones sloppily and then modify them when they fail. Critics allow this approach to work. Accepting the linearity assumption is HACKER’s way of avoiding combinatorial explosion of steps during initial planning; however, its detailed methods of categorizing errors and suggesting patches restricts its ability to patching plans that are almost correct already.

Another approach to the complications of, for example, conflicting subgoals has been pursued in systems that plan hierarchically, developing plans at successively lower levels of abstraction. NOAH (Sacerdoti, 1977) is perhaps the best known hierarchical system to accomplish tasks in the blocks world. Originally developed to help mechanics in assembly tasks, it maintains plans for action at a variety of levels of abstraction or generality.
Problems for NOAH are represented at a number of levels, and planning takes place from higher to successively deeper levels.

NOAH takes as input a problem and uses a variety of functions to decompose the goal of the problem into subgoals. This decomposition occurs recursively until the problem is represented at a level of abstraction on which actual problem-solving operators can operate directly. At each level of abstraction, the plan is checked to see if problematic interactions have arisen. If they have not, the planning process operates on a more detailed level. If they have, the help of one of a number of critics is employed to improve the plan as it becomes more elaborated.

In NOAH, the problem is represented by an initial world model containing the original state of the world when the problem was started. NOAH also possesses a structure called a TOME (table of multiple effects) for each problem containing a trace of the changes for every proposition that occurred in response to operator selection and execution. Thus, in NOAH, the original world model never changes, but becomes supplemented by a list of effects which have modified that original state.

NOAH maintains a library of critics like HACKER. A critic such as resolve conflicts looks at the addition and deletion lists (as represented by TOMICs) of subgoals, and if a delete list for one subgoal contains a precondition for another subgoal, a conflict is flagged. In one well-known example where this type of conflict arises, “paint the ceiling and paint the ladder,” painting the ladder deletes the precondition, has ladder, for painting the ceiling. By considering the add and delete lists, the anticipated interaction could be spotted before implementation. NOAH notices this and requires that the subgoal in danger of not being completed be attempted first.

Resolve double cross is a special kind of critic. Unlike resolve conflict, this critic does more than critique and reorganize a flawed plan. Resolve double cross must actually play an active role in problem solution and propose new steps in addition to the currently available plans. In the swapping-blocks task, which requires the use of resolve double cross, there are four blocks, A, B, C, and D, in two piles of two each. C is on top of A and D is on top of B. The goal of the task is to swap the two top blocks making (above D A) and (above C B). If NOAH immediately attempts the task without regard to future precondition violations, once one of these goals has been accomplished, the other will not be possible.

Although NOAH could begin again when it encounters a problem of this sort, it prefers instead to use the plan it has and make modifications to it. NOAH recognizes that first a block top must be cleaned, and then the other block put on top. Here, resolve double cross suggests this additional step, clearing, although it is in no way explicitly prescribed by the task description. NETWORK’s approach to these problems requiring special critics is discussed in Section 4.
PLEXUS (Alterman, 1988) is more appropriate to the discussion of case-based reasoners because it is capable of using and modifying old plans. However, its extensive use of background knowledge aligns it with NETWORK. In PLEXUS' scheme, background knowledge associated with a prestored plan is made explicit as a network. This background knowledge is not determined by any specific property of a plan, but rather by its position in the knowledge network. The adaptive planner refits old plans by interpreting the new situation and tries to find a piece of the network to match the current situation. PLEXUS is driven by this process of situation matching at various levels of a representation hierarchy. When a planning step fails, PLEXUS reverts to interpreting the situation at a higher level of abstraction, that is, as a class of actions, where a match may exist. Once a match at some level of abstraction has been found, PLEXUS proceeds back down the hierarchy until a specialized appropriate plan of action is found.

Like GPS and STRIPS, PLEXUS attempts to attain generality and planning flexibility, but unlike those systems, it considers much more background knowledge, and this knowledge may be used in fairly large chunks. PLEXUS exploits the structure of the knowledge and uses a plan's relative position in the knowledge network to evaluate its utility in the current situation. Because PLEXUS is knowledge based, its performance is dependent upon the accessibility of the structure and content of that knowledge. Hence, adaptive planning in PLEXUS, and (as will be shown) planning in NETWORK, depends upon the interaction of the systems' knowledge and the current task.

Although it appears that PLEXUS could be applied to domains other than everyday planning, it has not yet been shown how PLEXUS would solve problems of the sort typically attempted by planners. From a psychological point of view, it is just this ability to model performance in varying domains that gives a model theoretical power. The development of unifying themes in cognition should be a major goal of theoretical work and the construction-integration model provides a start towards this end.

The next section of this article contains a detailed description of the construction-integration model, followed by an illustration of the simulation of the model, and the problems for which it plans solutions. The article concludes with a discussion of NETWORK's performance and its relationship to other planners.

---

1 Wilensky (1983) proposed a hierarchical theory that suggests the use both of system knowledge and the current task situation. His theory is hierarchical in yet another way. Whereas, in Alterman's and other's work, tasks or situations are represented at a series of levels of abstractions, Wilensky's theory uses meta-planning knowledge. That is, his system has available to it knowledge about the planning process itself. This is quite different from the knowledge NETWORK requires, and thus, will not be discussed further here.
2. RATIONAL FOR AND ASSUMPTIONS OF THE CONSTRUCTION-INTEGRATION MODEL

NETWORK creates new plans for each task (although the idea of adding case-based reasoning is addressed in a subsequent section of this article), confronts problems of an intermediate level of difficulty, uses the existence of real-world knowledge, embodies no explicit knowledge of planning, and plans at a single level of abstraction. Most importantly, it departs from previous planning work in that it addresses the issue of planning as an instance of comprehension. Although systems exist that can perform both planning and understanding as distinct processes (e.g., Wilensky, 1981; Winograd, 1972), NETWORK produces plans as a response to understanding some set of instructions.

Our comprehension-based approach is appropriate for a particular class of tasks. NETWORK attempts to model solutions to tasks which are neither too easy, although novices still make errors on such problems, nor too difficult; experts appear to solve them effortlessly. This class of behaviors falls between two extremes. Fixed, precompiled responses will no longer suffice, but the deliberate character of real problem-solving is not evident either. Text comprehension provides a good example of this class: 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 just to disregard the difficulty and go on with the easy reading.

As Kintsch argued (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 (but, see Norvig, 1989, for an example of weak methods for text inferencing). The model proposed by Kintsch, 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 requiring little conscious control. Crude rules create a crude representation, however, and fine tuning is therefore necessary. This 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, 1989). 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 because it accounts for some empirical phenomena that are difficult to handle otherwise. It is also attractively simple, computationally.

It is this type of model 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.

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 both in 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 become a straitjacket when it cannot be adjusted with sufficient flexibility. 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, thus eliminating the need to modify a schema that never quite fits the specific context. Rather, a schema may be created on the spot, suitable for that specific context. That such a contextual generation of global schemata from the local relations in an associative network is possible under some special conditions has been shown by Rumelhart, Smolensky, McClelland, and Hinton (1986) and Kintsch and Mannes (1987). The work reported in the following extends this approach in the direction of
planning solutions to routine computing problems. In this formulation, it is assumed that the associative memory network contains all types of information: semantic knowledge of the domain, as well as knowledge of actions or commands that can be used to solve computing tasks. NETWORK's job is to use this memory to suggest a sequence of command selections, forming a plan for a task's solution.

One of the simpler tasks used during the development of NETWORK asked the system to produce a sequence of action selections that would "include an address that is known in a letter that is in a file, starting at the system level of the computer." It is referred to as the INCLUDE task. This task instruction is given to NETWORK in propositional form and it then attempts a solution. An associative network is constructed to represent the understanding of the task. This network contains many types of information, not the least of which is knowledge of the task to be done and related world knowledge. This network is integrated such that actions which are relevant and possible show high levels of activation, and irrelevant or currently unacceptable actions show little or no activation. For each step or solution action produced, NETWORK must complete an entire construction-integration cycle. As propositions representing the outcomes of each solution step or plan element are added to represent the newly changed hypothetical state of the world, integration produces new patterns of activation flow reflecting the changed state of the world.

For the INCLUDE task, NETWORK's first step in planning a solution is to find the file containing the letter being written. This is an understandable precondition for being able to edit the file in order to add the address. Once the location of the file has been found, NETWORK enters the editor, types the address, and exits the editor, leaving the file containing the letter changed to contain the address as well.

Routine computing tasks such as this one (i.e., tasks done in a semiautomatic manner on a regular basis), for expert users, belong to the intermediate class of problems discussed earlier. They cannot be solved with fixed scripts because, although each component of the task is highly familiar and scriptlike, 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 getting a file into the editor 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 new sequences and new variations. Some of these sequences might recur frequently enough and, as a result, be compiled into scripts themselves. This possibility has not been explored here. The current focus is on the more usual case where partly or entirely new action sequences must be generated to perform the assigned task.
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.

3. THE ROUTINE COMPUTING SIMULATION: NETWORK

Note that we are concerned with the production of plans. NETWORK does not perform the tasks it is given, it merely suggests a sequence of actions that would do so. The goal of the simulation is to understand brief texts that request a computer user to perform certain simple, routine computing tasks. Here, understanding means knowing what to do. More specifically, the simulation must propose a sequence 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 a selection of these to form a plan for the task it has been given. Thus, a plan for NETWORK is a proposed sequence of plan-element selections. NETWORK was developed on a set of tasks for which we had verbal protocol data, described in the next section, and then was tested on several new transfer tasks to test its generality and sensitivity to our parameter estimates.

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 consisting 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 selects, for the current plan step, the most highly activated plan element whose preconditions are met in the world. This proposed action changes the state of the world, and the whole process recycles, selecting another plan element, until it appears that the desired outcome would be

---

1 NETWORK is written in the CommonLISP programming language and documentation for its use is available (Mannes & Roushey, 1990). The documentation contains a description of NETWORK's low-level implementational details which are, for the most part, omitted here. Additional details may be found in Mannes, 1989.
achieved by the proposed sequence of plan elements. NETWORK's performance draws heavily upon the contents of its long-term memory. It is to the characterization and origin of this memory that we now turn.

3.1 The Long-Term Memory Network

Long-term memory is conceptualized as an associative network, the nodes of which are propositions (or concepts, which can be regarded as a type of proposition). Propositions were chosen as the basic unit for NETWORK because they allow information about the domain to be distributed in the system's memory (i.e., no one node represents a task description or a complete plan) and because of the psychological validity of propositions as the basic processing unit in text comprehension (e.g., 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 either to the magnitude or complexity of human memory. Instead, only a small set of nodes are used, which are more or less relevant to the tasks to be performed, and these are linked in a way that approximates some of what we know about 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, the simplified approach to simulating long-term memory suffices.

3.1.1 Contents of Long-Term Memory

3.1.1.1 Protocol Analyses. Although the associative network, which is NETWORK's long-term memory, is not all-encompassing semantically, its contents are based upon empirical data. To obtain the nodes for the simulated long-term memory network, a protocol study was performed in which six experienced computer users were given several routine computing tasks to perform while providing concurrent think-aloud protocols. Previous psychological research has shown that verbal reports are one effective way of investigating cognitive processes, particularly in the areas of planning and problem solving (Ericsson & Simon, 1984; R. Guindon, personal communication, January, 1988). All tasks involved routine actions such as handling and editing files and using the mail system. Subjects knew immediately what to do in terms of providing plans for accomplishing the tasks, and none were confused by the hypothetical situations with which they were presented. 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.

3.1.1.2 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). They were shown each proposition and asked to write down the first thing that came to mind. Their responses were propositionalized and included as additional nodes to the network. Subjects were uninformed as to the purpose of their participation and they often produced associations outside the computing realm. For example, in response to a proposition about mail, associations about the post office and stamp requirements were produced. Second, for all propositions that 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 requested tasks would be. Finally, a set of 26 plan elements were included in long-term memory. Each plan element corresponds to one basic operation or action that can be taken to accomplish a task. Most were produced by subjects, but some needed to be added so that NETWORK could attempt transfer tasks.

Plan elements are the basic actions the system has available. A complete list of plan-element names is given in Table 1 (p. 316). Planning in our simulation consists of putting these known plan elements together in the right sequence to produce the requested results.

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 that 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). See Figure 1 (p. 316) for a simple example. 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 being a single copy of the text in the buffer to being two copies of the
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?)

Plan-element name ———— PRINT FILE?
Plan-element precondition(s) ———— EXIST FILE?
Know FILE? LOCATION
Plan-element outcome ———— EXIST FILE? HARDCOPY

Figure 1. A sample plan element for printing a file.

text in the buffer. Similar outcome changes occur as editing- and file-manipulation operations take place. The implications of this are discussed later.

3.1.2 Structure and Connectivity in Long-Term Memory
In Section 3.1.1 we presented the origin of the contents of long term memory. Here, we describe its structure. All elements in the simulated long-term
memory are connected in a way intended to correspond roughly to the associative and semantic relations among the items. We have tried to approximate these relations by use of a metric called argument overlap: All propositions sharing a common argument are linked, as are propositions embedded in other propositions. For example, the following propositions share a reference to the concept *mail*:

\[
(P_1(USE\ STUDENTS\ MAIL))
\]
\[
(P_2(FROM\ MAIL\ JOHN))
\]

and as such, would be connected by a nonzero link in the network. The same is true of the following pair of propositions, which have the propositional embedding relationship,

\[
(P_3(REQUEST\ MESSAGE\ P_4))
\]
\[
(P_4(SEND\ YOU\ MANUSCRIPT)).
\]

Although this is a crude method, it has proven reasonably satisfactory in simulations of text comprehension and memory (e.g., Miller & Kintsch, 1980), and is simple and objective, computationally. It yields, however, only a rough approximation of the relatedness of propositions. Whereas 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 described in Section 3.2.

The long-term memory network, which is used in all solution attempts, has 81 nodes or propositions, represented as a single $81 \times 81$ matrix, each entry corresponding to the link strength between a pair of nodes. All rows, each representing a proposition’s relationship to all others, are normalized so that the values may be interpreted as probabilities in the subsequent knowledge-activation stage described later.

This long-term memory matrix is the general knowledge that NETWORK has of the to-be-performed tasks and the domain-related information. *It is created once and does not change.* Should we decide to address issues of learning, long-term memory would change with the addition of newly learned material. It might also change to admit propositions representing solutions to tasks as cases, as considered in Section 5.2.

When given a text, which is a request to perform a certain task, the simulation employs long-term memory to figure out what to do. Specifically, in understanding the task instructions and planning a solution for the requested task, NETWORK uses this general knowledge to elaborate and enrich the input it receives, and to propose a solution from the knowledge-enriched textual input. The following section shows how these knowledge enrichments are produced, and how the representation of the enriched-task text is derived.
3.2 Task Networks
The texts to be understood by NETWORK are always brief requests to perform some routine computing task. Texts are given to the system in propositional form (after Kintsch, 1974), categorized as INTHEWORLD or REQUEST propositions. An example of the INTHEWORLD proposition might be, [EXIST FILE: LETTER IN THEWORLD] to represent that “there is a file called letter in the world.” An example of a REQUEST proposition might be [REQUEST [READ YOU MESSAGE]]. This categorization is necessary for the following two reasons. With respect to the INTHEWORLD category, propositions given as part of a task description must be identifiable as such. Remember that long-term memory contains knowledge of many tasks and many actions. Although this knowledge becomes activated and plays a role in the solution process, it is necessary to distinguish this information from that which has been given. The request proposition must be identified as such so that NETWORK can select an appropriate outcome from long-term memory during knowledge activation. This process is described shortly.

3.2.1 Generating the Elements of a Task Network
Given this propositional input, the simulation uses its long-term memory as a source to activate knowledge enrichments including an outcome, related knowledge, and all plan elements. Activation of related knowledge is a probabilistic process, but outcome selection and plan-element activation are not. Knowledge activation is done once for each task as follows:

1. Given a REQUEST, NETWORK searches long-term memory for the corresponding OUTCOME, and adds it to the proposition list. This is accomplished using a modification of the Raaijmaker and Shiffrin (1981) search of associative memory model. In this model a composite cue is used to search memory, activating only items 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 ?I]. An intersecting search utilizes these propositions to select an appropriate OUTCOME proposition from long-term memory.

2. 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. For each proposition, n random probabilities, r_i, are generated and the row’s values are added successively until r_i is exceeded by this sum. These propositions, call them related knowledge, are also added to the proposition list.

3. All plan elements are bound to relevant objects in the world, and added to the proposition list. The plan elements in long-term memory have variable slots, such as FILE?, TEXT?, and so forth. This represents what NETWORK knows about files in general, for example, that they can be
ROUTINE COMPUTING TASKS

printed, deleted, and so on. Given an INthewORLD proposition such as [EXIST FILE 'LETTER INthewORLD], the variable FILE? in each plan element would be bound to FILE 'LETTER meaning that what is known about the class of files in general now applies to the particular case of a file, the 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.

Another complete set of plan elements, can-do plan elements, are created here. These are exactly the same in form as the original, want-to plan elements, but serve a unique role in the action-selection process. The function of these plan elements becomes clear in Section 3.2.4. These three types of knowledge enrichment and the original propositions provide the input for the next process (connectivity).

3.2.2 Connectivity in the Task Representation

There now exists a list of propositions containing three types of information (original task description plus its outcome, related knowledge, and plan elements) that will be used in proposing a selection of plan elements to compose a plan for the task’s solution. Thus, we have isolated the contents of the task network and now describe how NETWORK determines its structure. That is, how it assigns link strengths between all pairs of propositions. There are six specific cases, and a general default rule used to determine these:

1. The REQUEST proposition is linked to all plan elements with the same name as the request. 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 directly to any plan element, because there is no plan element named INCLUDE.)

2a. The OUTCOME proposition is linked to all plan elements that have the selected outcome. If the outcome of a request is that someone receives the file “letter,” this OUTCOME proposition is linked positively to all plan elements that have the outcome [RECEIVE SOMEONE FILE 'LETTER], that is, all SEND and REPLY-with-a-file plan elements.

b. Likewise, a plan element that would produce an outcome inconsistent with the requested outcome is given an inhibitory link to the OUTCOME proposition.

c. The OUTCOME has no link to plan elements having outcomes that deal with objects other than those represented in the OUTCOME proposition.
3. All plan elements that have all of their outcomes already existing in the world have inhibitory links to the corresponding INTHEWORLD proposition(s). For example, the proposition [KNOW FILE LETTER LOCATION INTHEWORLD] would have an inhibitory link to the plan element [FIND FILE LETTER].

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

4. If a plan element, A, requires a precondition X, plan element A 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 2.

5. If a plan element, B, requires precondition Y, it has an inhibitory link to all plan elements that have the outcome NOT-Y (i.e., would destroy condition Y in the world). An example is also shown in Figure 2.

6. The can-do plan elements are connected to the other propositions in a very straightforward manner. They are positively linked to their respective plan elements. They have no link to any other proposition except for inhibitory links to the INTHEWORLD propositions. In the situation where all of a plan element’s preconditions exist as INTHEWORLD propositions, the inhibitory links between the can-do plan element nodes and the INTHEWORLD proposition nodes are set to zero.

7. The final case for determining interrelations is a default case, creating a positive link among all propositions sharing a common argument. As previously discussed 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.

The precise link strengths for these cases are parameters to be estimated for the model, subject to certain general constraints. Inhibitory links must be made relatively strong numerically, because they must not be overwhelmed by the many positive connections in the network. Specific connections should be stronger than the default, argument overlap, connection. Where no link is specified, connections are set to zero. The original parameter estimates used were arrived at by trial and error while NETWORK was being developed. These same parameter values were used in attempting all transfer tasks.

Note that once connectivity is complete (i.e., all relationships have been identified), all information regarding the origin of the link values is lost. There are no link types, merely links of different strengths.

Informal evaluations of exact parameter values suggested that the actual parameter values used are not critical, although the relative magnitude of the values for the different types of relationships must be maintained in order for NETWORK to solve tasks appropriately.
Three plans in their generic (unbound) form

<table>
<thead>
<tr>
<th>NAME</th>
<th>PRECONDITION(S)</th>
<th>OUTCOME(S)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIND TILE</td>
<td>AT-LEVEL SYSTEM</td>
<td>(KNOW FILE LOCATION)</td>
</tr>
<tr>
<td>DCLCTC TILC</td>
<td>AT-LEVEL SYSTEM</td>
<td>NOT-EXIST FILE</td>
</tr>
<tr>
<td>PRINT TILE</td>
<td>AT-LEVEL SYSTEM</td>
<td>EXIST HARDCOPY FILE</td>
</tr>
</tbody>
</table>

TO A PLAN ELEMENT PROVIDING THAT SPECIFIC OUTCOME

<table>
<thead>
<tr>
<th>FIND</th>
<th>DELETE</th>
<th>PRINT</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIND</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>DELETE</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>PRINT</td>
<td>+</td>
<td>-</td>
</tr>
</tbody>
</table>

Figure 2. A subset of plan elements is shown in the top panel and the network interrelationships (causal chaining) between them are shown in the bottom panel.

It is important to note that although long-term memory is created once and forever for NETWORK, and knowledge activation is done once for each task, the links describing the task network are created dynamically for each step of a task. Because previous actions can change plan-element outcomes, a plan-element outcome 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 will be described later.

3.2.3 Integration of the Task Network

Through these operations, a connectivity matrix $C$, consisting of the relations between all pairs of all propositions, is constructed. Integration is accomplished via spreading activation. This is implemented by the vector-matrix multiplication scheme described next.

A vector, $A_1$, 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 text propositions (identifiable by their \textsc{intheworld} arguments) and the proposition representing the selected outcome are assigned a value of $1/k$, where $k$ is the total number of such propositions; all other elements (i.e., related 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 $A$ 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 for each proposition is less than some value $e$). 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 $A$, 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 $A$; for this, the distinction between can-do and want-to nodes needs to be considered.

3.2.4 Decision: Can-do and Want-to Nodes

Frequently, the most strongly activated plan element has conditions necessary for its execution that 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. 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 basing the plan-element selection on the can-do node activation.

Thus, all calculations take place in the network as previously described 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. As defined by their
ROUTINE COMPUTING TASKS

3.2.5 Adding Outcomes to the World: Planning Steps

At this time, the most highly activated can-do node has been identified. Its proposed execution will have a certain outcome. This outcome is represented by one or more propositions (specified in the outcome field of each plan-element—see Figure 1), which are now added to the network as INTHEWORLD propositions. These are things that would be true if the action were taken. The plan-element outcomes are added to the contents of the task network, the outcomes are compared to the outcome expected by the OUTCOME proposition, and the first step in composing a plan is complete. If a match is found between the expected OUTCOME and the current state of the world, the task is done; otherwise, the newly added information is linked to the already existing network by the same rules used to create the original network. Thus, a new expanded network is obtained, with somewhat different interrelationships among the nodes. Most importantly, the newly added INTHEWORLD propositions will now inhibit the plan element that was just executed (and any other that would produce the same outcome) by plan-world inhibition (Section 3.2.2, case 3).

The whole process of integration via spreading activation is now repeated with the modified network. As a result, a new can-do plan element will be selected, its anticipated outcome(s) will be added to the network as INTHE-
WORLD propositions, a check for task completion will be made, and if it fails, construction and integration will 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 incorporated into the plan.

At present, we are only concerned with a simulation of expert users and, thus, of correct solutions. Although data have been collected on novices performing the same tasks (Mannes & Hostetter, May, 1989), and they certainly make errors which must be accounted for, an error theory is not yet part of our model. Thus, we include no processes such as backtracking. At this point, backtracking would not be feasible because, in contrast to systems like NOAH, which maintain information about the original problem state, once a plan element has been selected and its outcome added to the world, NETWORK has no idea about what just happened. It only knows what the world looks like now, without regard to how it became that way. The following section explains how NETWORK uses the processes just described to understand and plan solutions to a subset of the specific tasks it has been given.

4. UNDERSTANDING TASKS

4.1 The INCLUDE Task
At the time a task is encountered, long-term memory has already been created. NETWORK's job is to add knowledge enrichments from long-term memory to the description of the task it has been given, and select, on the basis of construction-integration cycles, a plan element for as many steps as the task requires.

4.1.1 Task-Network Contents

4.1.1.1 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 nontrivial one. Several inferences requiring situational knowledge have been made. This aspect of the problem is not dealt with in our model at present. 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 that do not have the same status.

4.1.1.2 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 assigns X to ADDRESS and Y to LETTER. To produce this outcome, [IN TEXT ADDRESS FILE LETTERADDRESS 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. For INCLUDE, only four associates were generated in this way 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 future 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 5. Third, the whole set of plan elements (and their newly created can-do plan-element duplicates) is selected from long-term memory and used in the subsequent binding process, described next.

4.1.1.3 Task-Appropriate Plan Elements. The variables contained in the plan elements available in long-term memory 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.

4.1.2 Task-Network Connectivity

The origin of the task-network contents has now been described. They are linked to form a connectivity matrix by using the methods described in

---

1 The number 2 is arbitrary; in various applications we have tried values between 2 and 10.
Section 3.2.2; REQUEST and OUTCOME propositions are linked to the plan elements, plan-world inhibition is calculated, the causal chain among plan elements is derived, and all cases of argument overlap are identified.

4.1.3 Integration and Decision Cycles
The strength of relationships among all 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 pairs of plan elements form the entries of a 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, 0.4. Links among plan elements, which provide the causal chaining, were assigned a value of 0.7, 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 A1 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. A1 was postmultiplied by C 9 times and normalized after each multiplication, until the average change in activation values was less than .0001.

Figure 3 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 the original want-to plan-element 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 is proposed as the action to be taken for the first step of INCLUDE and, as a consequence, a proposition stating that the location of the FILE LETTER is now known and will be added to the INTHEWORLD list and linked to the task network.

A new integration cycle begins in which the initial activation vector for Step 2, A2, 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 4, p. 328). However, the third-strongest plan element, [EDIT FILE LETTER], now does have its precondition satisfied—we know where the file is—the corresponding can-do plan element is activated and the system therefore proposes to edit the file as the second step. This step, once again, changes the state of the world. Not only does
it change the level at which NETWORK is operating, but it 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. They are added to the task network and linked accordingly.

On the third integration cycle NETWORK (Figure 5, p. 329), although it still cannot act on its first choice because the TEXT ADDRESS is not in a buffer that could be pasted into FILE LETTER (a precondition for PASTE), selects [TYPE TEXT ADDRESS]. The outcome of this step is [IN TEXT ADDRESS BUFFER LETTERADDRESS INTHEWORLD], which is close to what we want (i.e., [IN TEXT ADDRESS FILE LETTERADDRESS INTHEWORLD]), but not quite right yet.
At this time, the outcome of the [EXIT FILE] plan element 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 [IN TEXT ADDRESS FILE LETTERADDRESS IN THE WORLD], the selected outcome for the specified request. This allows it to receive a link from the strongly activated OUTCOME proposition via Case 2 of Section 3.2.2 and permits NETWORK to distinguish between quitting and exiting the editor, a distinction that was nonmeaningful 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 \texttt{EXIT EDITOR} plan element being most active as shown in Figure 6 (p. 330). 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 \texttt{[IN TEXT^ADDRESS BUFFER^LETTERADDRESS]} into \texttt{[IN TEXT^ADDRESS FILE^LETTERADDRESS]}, and this matches the outcome of the original request to include the address in the letter. The task is solved; the instructions have been understood and a plan has been proposed. We now show how \textsc{Network} attempts a well-known task using the same procedures and parameters as for \texttt{INCLUDE}. 

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{activation_values.png}
\caption{Activation values for all plan elements for the third step of the \texttt{INCLUDE} task. Decisions are based on the can-do values.}
\end{figure}
4.2 The PRINT-AND-DELETE (Conflicting Subgoals) Task

As mentioned in the introduction, conflict resolution has long been of interest in the planning literature. 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 separate the compound task into its constituent 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 previously mentioned work of Sussman (1975), to take task-subpart interdependencies into account. In NETWORK's domain, "Print and delete the file eggplant" is an analog to the traditional painting problem.

NETWORK was given instructions to print and delete the file eggplant. It enriched this task description by activating some long-term memory propositions and all plan elements. A task network was constructed and integrated; Figure 7 shows how NETWORK handled this situation. For brevity, activa-
Figure 7. Activation values for all plan elements for the three steps of the PRINT/DELETE task.
tion values for only a subset of relevant plan elements are shown. In the first step, PRINT is most highly activated among the want-to nodes (although its preconditions are not met), but not DELETE, because it is inhibited by PRINT through the causal chaining in the network (deleting the file would remove a precondition for printing it just as it would for finding it, see Figure 2). 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 INTHETWORLD, deletes it in the third step. Here, the causal chaining has done the work that typically required resolution procedures. Rather than creating a sloppy plan and then fixing it, NETWORK gets it right from the start.

The causal chaining is not solely responsible for NETWORK's apparently intelligent behavior. In this example, spreading activation was essential to ensure that the solution would be formed properly. If the activation values for the can-do plan elements were examined immediately after the file had been found, one would have found the following situation. The two most highly activated can-do plan elements are PRINT and DELETE. Their activation values are identical as they are both direct requests and both produce the desired outcomes. Barring any complicated selection procedure and no spreading activation, one of the two would be chosen randomly, leading NETWORK to begin forming a plan for the task, which 50% of the time, would be unsolvable. Thus, NETWORK achieves, through causal chaining and spreading activation, the type of behavior that required a special critic—resolve conflict—in NOAH. The function of another special critic, resolve double cross, can also be achieved by NETWORK's methods, as the following describes.

4.3 The RENAME (Swapping Blocks) Task
An analog to swapping blocks was designed for our computing domain. NETWORK was told to rename file^a file^b and rename file^b file^a, given that it already knew their locations. The appropriate knowledge-enrichment processes were conducted and NETWORK approached the problem in the following manner. After the first integration cycle, two plan elements were tied with the highest activation. They were not the RENAME plan elements as might be suspected. These had little activation because they inhibited each other. The selection of one would make the other impossible. The most highly activated plan elements were those to duplicate each of the files. One, DUPLICATE FILE A, was chosen at random, and this produced a copy of the contents of file^a and the creation of many new plan elements to deal with this newly created file. This allowed the OUTCOME proposition to plan-element links to reflect the fact that now file^b could be renamed to file^a and the contents of file^a would still be maintained in the duplicate file. This was done and the final step chosen was to rename the duplicate file^a to file^b.
In this case, NETWORK was able to solve the task because so many of the other plan-element outcomes were inconsistent with the goal of the task and, therefore, the outcome to plan-element links played a strong role.

4.4 The SEND Task
NETWORK has shown other surprising behaviors as well. When told that it has just received and read a mail message from a colleague asking for a paragraph from a paper it has on file, NETWORK produces an orderly sequence of plan-element selections. It first exits the mail system, it finds the file, enters the editor, copies the paragraph, pastes the paragraph into another buffer, and exits the editor. Because NETWORK is equipped with a plan element to send files at the system level, this is what it was anticipated to do. Surprisingly, NETWORK chose to reenter the mail system and REPLY to the message it received rather than SEND the file at the system level. Cognitively this makes sense, why type in the address of your colleague if you needn't do so. It would be absurd to claim that NETWORK has thought of this. The obvious explanation centers on the fact that mail messages were mentioned in the task description and, due to the links produced by argument overlap, the REPLY-IN MAIL plan element receives slightly more activation than the one to send the file at the system level. Consider the next task as a contrast.

4.5 The REVISE Task
The REVISE task is to “revise a manuscript you are working on with a colleague by removing a paragraph, and then send the revised manuscript to that person.” Past research (Kintsch & Mannes, 1987) has shown that subjects providing verbal protocols of script actions, for example, “buying groceries,” employed special linguistic signals such as “and then” to mark the major boundaries between script units. It was, 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, almost as two separate tasks, to be performed in sequence.

That is how NETWORK approaches this task. It is given the task instructions in two separate chunks and it plans a solution for the instructions up to the “and then,” revising 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 plans the remainder of the solution. 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. 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. The lack of this
information in the current task precludes NETWORK from trying to use the mail system.

5. DISCUSSION

What does NETWORK do for us that other planners have not already done? NETWORK appears to plan solutions to routine tasks. It does so in a manner which is rather different than many other systems. For one thing, NETWORK and the construction-integration model on which it is based make extensive use of the system's long-term memory. Many other planners have tried to minimize knowledge of anything other than the specific domain within which it plans solutions.

5.1 Use of Long-Term Memory

NETWORK is based upon a model of text comprehension. Any task which is attempted must first be understood. When NETWORK is presented with a task description, its first order of business is to activate related information in long-term memory. It may be that NETWORK has solved the tasks presented here without the benefit of using the world knowledge that was activated in long-term memory, but these tasks are without ambiguity. When NETWORK's solutions to tasks that have several different interpretations are examined, the use of this knowledge becomes apparent. In some cases, the activation of this knowledge could keep NETWORK from attempting to solve tasks in silly ways. Here is a hypothetical example.

Assume that NETWORK has been extended to deal not only with the computer domain which is of interest now, but also with routine tasks in everyday life. That means it has available to it plan elements and world knowledge about things like making breakfast, driving a car, and mailing letters. When instructions are received that mention the receipt of a mail message asking for some part of a manuscript file, knowledge in long-term memory which is related to mail, both computer and postal, messages, manuscripts, and files becomes active. When activation is spread throughout the task-connectivity network, interrelated items—those concerned with the computer interpretation of mail—all activate each other. NETWORK need not activate information in a directed manner or draw in just the appropriate information from long-term memory. Much information becomes activated, and integration weeds out the irrelevant information. Here, there would be little support for the plan elements dealing with regular mail, and by the time the network has settled, these plan elements would have become inactive or at least overshadowed by the activation of the electronic mail plan elements. Thus, NETWORK has used long-term memory to disambiguate the intended meaning of "mail." So, it seems that, unlike many planners where scaling up may harm performance, in theory, adding...
more information to NETWORK's long-term memory might improve its performance, computational issues aside.

Consider another example, where NETWORK has been instructed to add some information to a file that already exists. It may choose to do so in a variety of ways. Assume that NETWORK has been asked to include the discussion from one manuscript into the introduction section of another manuscript. It has been told that both of these papers exist in files on the computer. The solution produced will depend upon the knowledge-activation stage. There exist, in long-term memory, propositions regarding the length of various types of text, and their characterizations (e.g., ISA SECTION LONG TEXT). The propositions making statements about length include arguments which enable them to discourage various plan elements. In this case, knowing that any type of text is long will discourage the plan elements for typing, whereas a proposition stating that (ISA ADDRESS SHORT TEXT) will not. If NETWORK activates information about the length of texts which are sections, as required in this task, then this information will serve to inhibit the use of the TYPE plan element. Because this plan element accomplishes the same result as the PASTE plan element, PASTE is preferred, as long as the text to be pasted already exists in a file. Here, NETWORK has used the knowledge it maintains in long-term memory to select between two possible ways of accomplishing the same thing. Note that because knowledge activation is a probabilistic process, the information about text length will only be activated some proportion of the time. Thus, the solutions NETWORK produces are not determined, but depend upon what is currently active in long-term memory.

At first glance, this nondeterminism may not seem important. It is only after careful consideration of the verbal protocols and subsequent subject interviews that this becomes essential in simulating human planning. After producing an extremely unparsimonious, though successful, solution to one of our tasks, subject B was asked why he hadn't described a much simpler way of accomplishing the same thing. Because of personal experience with the subject, the experimenter knew that B possessed the knowledge that would have enabled him to produce this more parsimonious sequence of actions. His reply was simply that he hadn't thought of it. So it seems that even the most knowledgeable and experienced users sometimes (more often than is realized perhaps) produce less than optimal solutions for fairly routine and simple tasks. This is one way in which the model is an instance of comprehension. Nothing is determined, and depending upon the current "mindset" of the system, the solutions to the same task may differ.

Because all memory in the construction-integration framework, including episodic memory, is considered to be associative in nature, there is no reason, in theory, why information about previous problem-solving episodes should not be available to NETWORK in addition to the semantic memory
contents, the use of which has already been described. This episodic knowledge could be activated during knowledge enrichment and could also play a role in shaping NETWORK's plans. In fact, preliminary investigations of the introduction of cases to NETWORK have been carried out.

5.2 Case-Based Reasoning
In our attempt to use the notion of cases, we chose an arbitrary, propositional representation for NETWORK's cases. Each case, or episodic memory of having solved a task in the past, was represented by a single proposition. This proposition contained all of the arguments from the three most highly active propositions at the time the task was solved and the names of the plan elements that had been selected during the task solution.

For example, the memory trace formed as a result of solving the INCLUDE problem (Section 4.1) consists of the episode name, the six arguments of the three most strongly activated nonplan-element propositions, plus the names of 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 more strongly to any proposition or plan element containing FILE LETTER, resulting in a link strength of 1.2 instead of 0.4 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, if this case is activated in long-term memory during knowledge enrichment. (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.)

Figure 8 shows the performance of NETWORK on the first step of the REVISE task (Section 4.5), with and without memory for cases. The only memory for cases NETWORK had at the point this investigation took place 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 3.2.1): 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 file and dealing with a manuscript. Because these remembered cases are linked strongly both to the text propositions and plan elements, they become strongly activated Appropriately the REVISEI 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 increase further the strength of relevant plan elements, like FIND, and decrease that of irrelevant ones (like READ and FIND MESSAGE).\(^4\) On the other hand, the irrelevant plan element [PASTE TEXT TO 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.

This is a quite different model than most other approaches to case-based reasoning. In those systems, problem solving is often accomplished in the following manner. First, memory is searched for an appropriate previous

\(^4\) Also, due to some stronger links than before, the network might settle faster after spreading activation, which could be used as a measure of planning efficiency.
case. In Kolodner’s work (1984; Kolodner & Simpson, 1989), indexing appropriate cases is done by organizing cases based upon generalized episodes. Individual cases are indexed by the features they display which differentiate them from the others in their class. Although this generalized categorization effectively limits the search of memory, it does require the development of an effective generalizing mechanism. Then, there is no guarantee that only one case will be selected. If several are selected, some sort of evaluation procedure must be instituted to help select the best one. Often, this process entails the simulated use of the plan suggested by each case, and the comparison of the hypothesized outcomes with those expected by the goal of the task. This may be a costly process if there are many apparently relevant cases or if the plans involved are long. The case having the best evaluation becomes abstracted or generalized and then the system follows it in a step-by-step manner until some failure occurs. A search for similar past failures takes place and the selected plan is modified to bring the case under consideration closer to the current situation, or another case is selected as best.

To plan in real time however, a different approach that attends more to the execution and interpretation of plans rather than to the construction of plans, and treats planning and acting as inseparable, is preferable. In the purest implementation of this paradigm, “the planner will end up working from reflex-like stored responses for each immediate situation rather than true goal-directed plans” (Marks, Hammond, & Converse, 1988, p. 274). This is much more in the spirit of the construction-integration model than traditional case-based reasoning approaches. In NETWORK, there are no case-selection processes, no case evaluation, and no modification of plans by analogy, and so forth. 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 a situation where there is 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.¹

¹ We presume, however, that given a richer episodic memory, fatal interference effects could arise in this way. Of course, because we are concerned with a simulation of how real people solve these routine problems, similar interference effects can probably be observed in people, too.
5.3 Planning and Acting

In addition to the ways in which planners differ (mentioned in the introduction), planners also differ in another major way. Most plan entire solutions before implementing them and others plan a step at a time, interleaving planning and acting as recommended by McDermott (1978). NETWORK's networks are configured dynamically. The network representing the state of the world changes after each step to take into account the changes that have occurred in the world. 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 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 that drives the action (Greeno, 1989; Suchman, 1987). Although other planners achieve similar results by something like reconfiguring add and delete lists, this responsiveness to the dynamics of a situation is part of the basic architecture of our system, and the effects of future steps needn't be guessed about prior to that action taking place.

NETWORK reacts to changes in the situation—somewhat like persons who have acquired some routine with the Tower of Hanoi puzzles 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 might 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.

6. CONCLUSION

NETWORK does some interesting things, in interesting ways. It is not meant to be an artificial intelligence program to help 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 that 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. 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.

An exploratory work like NETWORK probably raises more questions than answers. Psychological research on memory for cases might put the use of case-based reasoning in NETWORK on a more stable foundation. In addition, to make NETWORK even more reactive to its environment, a richer representation of that environment and its nonpropositional 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.

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. (1989) 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.

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 in human cognition.

REFERENCES

ROUTINE COMPUTING TASKS


