Action Planning: Routine Computing Tasks

Suzanne M. Mannes & Walter Kintsch

University of Colorado, Boulder

The tasks we are concerned with here are routine operations involving a file and mail system. Our subjects are experienced computer users, for whom such tasks present no challenge. However, solutions for these tasks cannot be precompiled, because while the elements of each task may be familiar, and indeed overlearned, they are often put together in novel sequences. In other words, we are not dealing with fixed scripts which only need to be retrieved from memory, but with plans for action that are generated apparently effortlessly in the context of each specific task. The question addressed here is how these plans are generated.

THEORETICAL BACKGROUND

The problem statement for each task is a piece of discourse, usually a sentence or two with which the experimenter instructs the subject what to do. (More generally, one can assume that computer users instruct themselves in similar way in the course of their activities). The theory of discourse understanding, developed in a quite different context by Kintsch & van Dijk (1978), van Dijk & Kintsch (1983), and Kintsch (1988), will be used to model the way in which subjects comprehend this discourse.

We are interested in how the instructional text activates the knowledge a subject has about the tasks involved, resulting in a well-formulated plan for action. We are not studying the actions themselves: this is probably best done within a GOMS- or CCT-like framework (Card, Moran, & Newell, 1983, Kieras & Polson, 1985). What we do here is generate the GOAL statement, which is required to activate the high-level productions in these models. Once you have a PLAN – in the sense discussed below – you can then activate a sequence of productions for executing that plan, in the manner described by these models.

Understanding an instructional text means coming up with an appropriate plan for action. The approach we take to modelling this process is analogous to our previous work on word arithmetic (Kintsch & Greeno, 1985). Understanding a word arithmetic problem means generating a mental representation of the sets and set relations involved, on the basis of which the correct computations can be performed. This is achieved by activating both relevant general world knowledge and specific knowledge about arithmetic. Analogously, in the present case, we need to activate knowledge about the general nature of the task we want to perform, as well as knowledge about the specific computer system on which we are working. Thus, the same general theory of comprehension is being applied to two different domains.

PROTOCOL ANALYSES

Three experienced computer users were given the following three tasks with standard think-aloud instructions:

(a) *Include an address that you know into a file on the computer.*

(b) *You got a manuscript from someone and you want to edit it, making modifications, and then return it to the person you got it from.*

(c) *You got a message from someone while in mail asking for a paragraph out of a manuscript of yours to be sent to them.*

Their actions as well as their verbal protocols were recorded and analyzed by means of a method described in detail in Kintsch & Mannes (1987). The resulting temporal sequences of action planning statements and elaborations provided the starting point for our analyses. They gave us

some idea about the sort of knowledge that is activated when subjects perform these tasks, as well as about the sequence of planned and actual actions involved. Table 1 shows a greatly abbreviated summary of the verbalizations and actions of Subject 1 performing Task b.

We note the evidence for backward reasoning in Table 1, which is typical for these protocols. It is the anticipation of things to come which often drives current actions. In Table 1, the subject recognizes that in order to be able eventually to send the revised file, she must first enter the mail system to get the file[†]. Our theoretical goal will be to simulate the sequence of planned and actual actions, as observed in our data (but not the accompanying verbalizations).

THE MODEL

The main components of the construction - integration model proposed by Kintsch (1988) are a propositional textbase derived from the instructional text, an associative long-term memory, a knowledge activation process by means of which the textbase propositions activate information from the long-term memory, and, following these construction phases, an integration process which selects what fits together, and deactivates what is contextually irrelevant. Furthermore, these processes are cyclic, because the text is processed in sentence-like units rather than as a whole, and because when some action is performed, the state of the world changes. Thus, processing is repeated until the desired change in the state of the world has been achieved, or until it fails.

Long-term Memory

Forty-two propositions were taken from the protocols of subjects performing the three tasks described above to simulate a portion of a user's long-term memory network. Interconnections in this net were computed by finding all the instances where a given proposition shared an argument with another proposition, and where it was embedded in another proposition. Additive connection values of .5 were used for each case of argument overlap or embedding. A LISP program computes the interconnection values among all the items in long-term memory, creating a matrix of size n x n, called the Connectivity Matrix. This matrix approximates an association matrix: since we do not know the actual associative strengths among the propositions in long-term memory, we use this crude and fallible, but relatively simple and objective method to estimate these values. The second portion of the simulated long-term memory consists of 15 PLANS, as shown in Table 2.

Each plan has three fields, a plan name, preconditions, and outcomes. Preconditions correspond to propositions designating states of the world which must be fulfilled in the environment for the plan to be executable. Outcomes correspond to propositions which become

Table 1. Abbreviated Protocol: Subject 1, Task b (REVISE).

GET INTO MAIL	TO GET TO THE FILE
SAVE MESSAGE TO FILE	ASKS FOR FILE NAME
EXIT MAIL	
EDIT FILE	I'D BE IN EMACS WITH MANUSCRIPT FILE
EXIT EDITOR	
SEND MAIL MESSAGE	ASKS WHO TO SEND TO

1† The verbal protocols provide other instances of this reasoning, e.g.:
First I'd send my method section of the paper. I'd get into the editor one way or another and – first I'd have to look at the paper you are talking about. So I would have to think through how I stored that in my directory structure... So I'd have to think through how I have organized my stuff and go find the paper and then bring it up in the editor. I'd read your letter and then I wouldn't want to reply right away. I'd want to go up – you know – figure out what was going on.

states of the environment as a consequence of executing a plan. Plan names are propositions, thus consisting of a predicate and a number of arguments. Sets of plans, called fields, may be mutually exclusive (e.g. the four ways to send mail in Table 2). In such fields, the most strongly activated plan inhibits the other members.

Connections between the propositions in the long-term memory net and the plans are computed by the same algorithm based on argument overlap and embeddings, based upon the plan name only. An exception must be made, however, for certain types of proposition which we call REQUESTS and OUTCOMES.

Requests are the imperative verbs that are used in problem statements (requesting the user to do something - edit, send, etc). The relationships between these verbs and plans has to be more precise than the one afforded by argument overlap: each particular request must be associated directly with an appropriate plan. Thus, the REQUEST *EDIT* is tied to the plan (EDIT FILE) by a value of +1, and has 0 connection strengths with all other plans. Each OUTCOME proposition is similarly connected by a value of +1 to plans that produce this outcome, by a value of -1 to plans which produce an incompatible outcome but which include task relevant objects, and by a value of 0 to plans which produce irrelevant outcomes (because they deal with objects that are not involved in the task at hand). Only ultimate goals are connected in this way: other plans which need to fire before the ultimate plan can be accomplished must be contextually activated through the network as a whole.

TEXTBASES AND SITUATION MODELS

From the text of a problem (such as the three tasks mentioned above) a propositional textbase is derived by handcoding, in the manner of Kintsch (1985). Each text proposition then activates two other propositions which are associated with it in the long-term memory net. This is done by computing the argument overlap and embedding relations between that proposition and all other propositions in long-term memory and normalizing this vector (so that the sum of all interconnections is 1).The resulting connection strengths values can then be treated as probabilities,and two associates of each propositions can be sampled with replacement.

Table 2. List of PLANS.

Plan Name	Preconditions	Outcomes
(COPY TEXT FILE)	(@SYS) (IN FL DIR)	(IN TXT FL) (IN FL DIR)
(COPY FILE)	(@SYS) (IN FL DIR)	(IN FL DIR)
(READ FILE)	(@SYS) (IN FL DIR)	(READ FL)
(EDIT FILE)	(@SYS) (IN TXT FL) (IN FL DIR)	(IN TXT FL) (IN FL DIR)
(FIND FILE)	(@SYSTEM)	(IN FL DIR)
(SEND TXT FL ML)	(@ML) (IN TXT FL) (IN FL DIR)(IN MSG ML)	(RECV TXT)
(SEND TXT MAIL)	(@ML) (IN MSG ML)	(RECV TXT)
(SEND TXT FL SYS)	(@SYS) (IN TXT FL) (IN FL DIR)	(RECV TXT)
(SEND TXT SYS)	(@SYS)	(RECV TXT)
(COPY MESSAGE)	(@ML) (IN MSG ML)	(IN TXT FL) (IN FL DIR)
(READ MESSAGE)	(@ML) (IN MSG ML)	(READ MSG)
(FIND MESSAGE)	(@ML)	(IN MSG ML)
(SAVE MESSAGE)	(@ML) (IN MSG ML)	(IN TXT FL) (IN FL DIR)
(ENTER SYSTEM)	(@MAIL)	(@SYS)
(ENTER MAIL)	(@SYSTEM)	(@ML)

Once again, request propositions require special treatment: they do not randomly activate associated knowledge, but must be used to retrieve an expected outcome. Specifically, a proposition of the form (REQUEST X (Y)) together with (OUTCOME $) is used as a joint retrieval cue to retrieve W(Z) - the outcome of doing X(Y) which is associated to both retrieval cues, as in Raaijmakers & Shiffrin (1981) and Kintsch & Mannes (1987). Thus, we now have text propositions, plus their associates and outcomes. To these we add all 15 plans: when trying to solve a problem of the type considered here, all plan-knowledge must at least have the potential to be activated.

The interconnections among the nodes in the resulting matrix are computed in the same way as in the original long-term memory matrix. There are, however, two further complications, concerning the relations between outcomes and plans, and among the plans themselves. If all of the outcome propositions for a plan already exist in the models current state, they inhibit the plan that produces these outcomes. Thus, if the location of FILE-X is already known, the plan (FIND FILE-X) is inhibited. Expected outcomes of plans, on the other hand, are connected to other plans in the same way as in the long-term memory matrix, i.e. they activate plans that produce these outcomes, and inhibit plans that produce incompatible outcomes. Plans themselves are interconnected by a causal chaining mechanism. That is, a plan that requires precondition X activates all plans that have X as an outcome. Thus, the interconnections among plans represent the user's knowledge about the causal relations in the system.

In this manner, a connectivity matrix of size n x n is obtained for each text, corresponding to the original m text propositions and to the (n-m) associates, outcomes, and plans that have been added to the text. An initial activation vector with n elements is then constructed, with activation values 1/m for the m original text propositions and 0 for everything else. This vector is then repeatedly postmultiplied with the connectivity matrix, and renormalized after each multiplication, to compute the spread of activation among the elements of the vector. When the activation vector reaches an arbitrary criterion, such as an average change after a multiplication of less than .0001, the activation pattern is considered to reflect the stable situation model formed by the user, given the particular task and the knowledge assumed here.

In terms of the van Dijk & Kintsch (1983) model, we have started with a small *textbase*, which activated both relevant and irrelevant *knowledge*. We then used an *integration* process to deactivate the irrelevant knowledge that had been introduced, thus arriving at a *situation model* that corresponds to the user's understanding of the situation and the task to be performed.

Plans and Action

Plans end up at different levels of activation in the situation model. We postulate an executive process that checks the plan with the highest activation level and executes it if its preconditions are satisfied. If they are not, it goes to the next highly activated plan, and so on. If the plan executed has as its outcome the expected outcome (which was retrieved from the long-term memory by means of the original request), the problem is solved. If not, the consequences of the plan that was executed are added to the situation model, and the integration process is restarted, resulting in a new pattern of plan activation. This process is repeated until the desired outcome results (or is stopped when it goes hopelessly awry).

THREE EXAMPLES

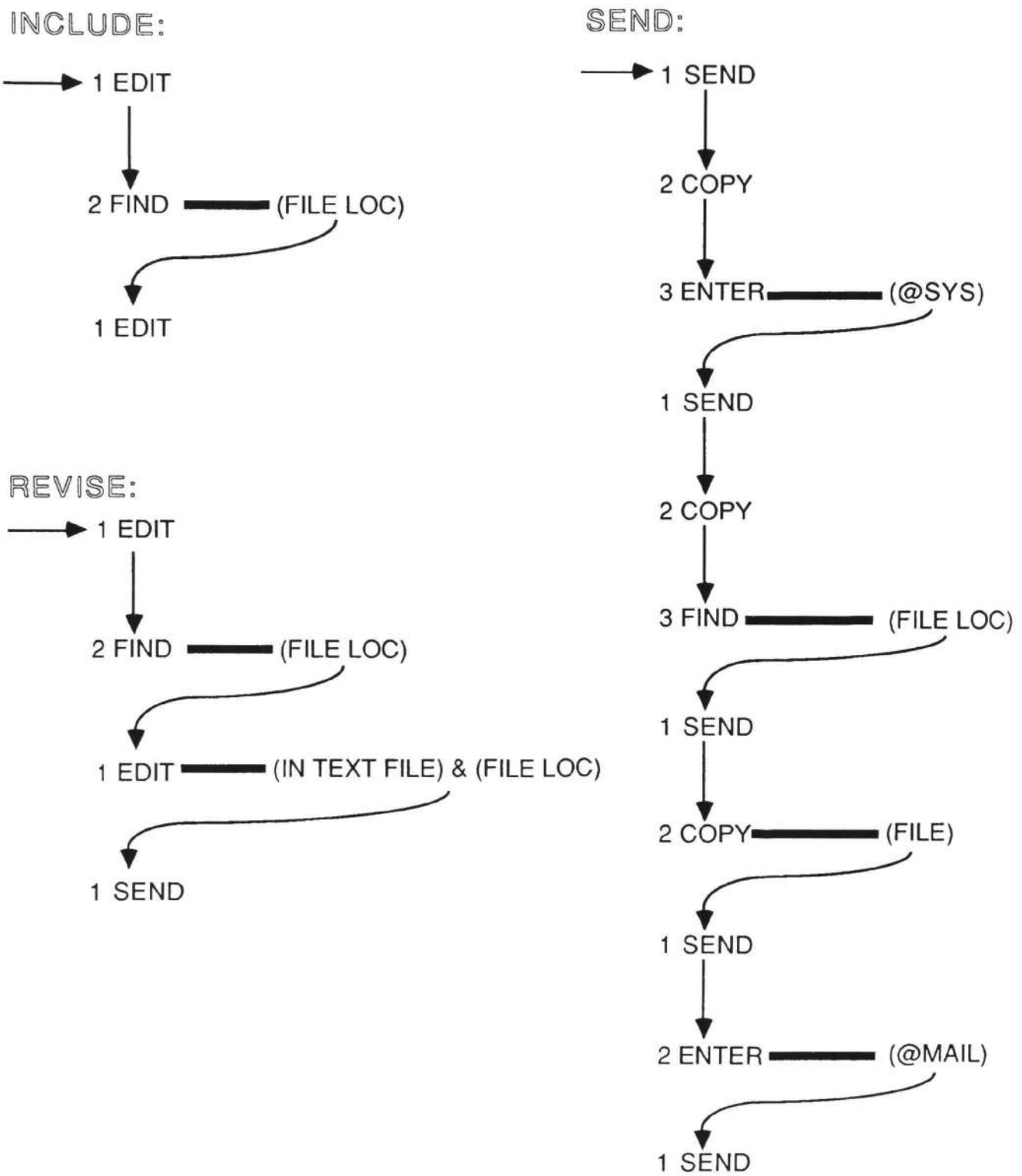
The results of a simulation with this model of the three problems shown above are partially depicted in Table 3. Only the plans actually considered by the executive process are shown, together with rankings indicating their relative activation values. For the *INCLUDE* problem, the (EDIT FILE-LETTER) plan receives the highest activation value in the first round. It cannot be executed because one of its preconditions is not fulfilled: the location of FILE-LETTER is unknown. Therefore, the second-ranked plan is considered. This happens to be (FIND FILE-LETTER), it is executed, and results in the addition of the proposition (KNOW FILE-LETTER LOCATION) to the situation model. After a new integration phase, the (EDIT FILE-LETTER) plan again is the most highly activated one. It is now possible to execute this plan, which provides

the desired outcome (IN TEXT-ADDRESS FILE-LETTER), completing the problem. The *REVISE* example is segmented into two components by the linguistic cue *and then*: such temporal connectives tend to be used as signals of episode boundaries, and hence as segmentation cues (Kintsch & Mannes, 1987). The request which is acted upon first is to *revise the manuscript*: the model wants to EDIT, but cannot do it, so its executes the next highest plan, which results in a new proposition characterizing the state of the system, (KNOW FILE-MANUSCRIPT LOCATION). After a new integration process, EDIT is still at the top, but now it is possible to execute this plan. As a result, the propositions (IN TEXT-MANUSCRIPT FILE-MANUSCRIPT) & (KNOW FILE-MANUSCRIPT LOCATION) get added to the textbase, the request to revise is dropped, and the request to send it is substituted. The integration process selects the plan (SEND TEXT-MANUSCRIPT FILE-MANUSCRIPT @SYSTEM), all of its preconditions are met, the plan is executed and the problem is solved. The *SEND* problem is somewhat more complex and requires five construction-integration cycles before the solution is arrived at. From the very beginning, the model wants to SEND, but is unable to do so. All it can do in the first cycle is to leave the MAIL system. In the next cycle, it manages to locate the file it needs, so that it can copy the to be sent text into a FILE-TEXT. However, in order to send this file, it wants first to re-enter the mail system, because it prefers to reply to the message it received, rather than sending a new one as in *REVISE*.

CASE-BASED REASONING

In the simulations described above, the program used only its general knowledge about plans in arriving at solutions. What would happen, however, if these problems, or similar ones, were presented for a second time? Although we don't have any direct empirical evidence as yet, we would expect to see examples of case-based reasoning. It appears plausible that the to-be-comprehended text would remind subjects of having solved this problem, or a similar one, before. There are many ways in which a case-based reasoning capability could be added to the present simulation. As an initial exploration, we decided to make some simple assumptions about memory for an episode. Specifically, we assumed that what would be remembered from an episode would be the three most highly activated text propositions, together with the plans that were executed in performing the task. Thus, four cases, each consisting of the three propositions and the corresponding plans, were added to our long-term memory. (There were four cases, because the Revise problem consisted of two episodes). The three problems were then re-run. In each instance, the text propositions activated the appropriate earlier case, and sometimes even other cases. Thus, the Revise text reminded the system not only of the earlier Revise episodes, but also of the Send episode, which also involved a text and file called *manuscript*. In the comprehension network, each case was connected only to the plans that were actually executed in this episode. Not surprisingly, this had the effect of further strengthening these plans, and all problems were solved correctly, as before. Substantial portions of the activation of each plan, however, derived from its connection to earlier episodes: between 40% and 52% of the total activation of non-ultimate plans (i.e. plans that had to be executed before the ultimate goal could be achieved) was acquired from remindings of previous cases. For ultimate plans, this proportion was lower, ranging between 20% and 29%. Thus, cases helped particularly in the early part of the comprehension process, while the last step is well defined by the text anyway.

Table 3. Traces of solutions to the INCLUDE, REVISE, and SEND tasks.



CONCLUSIONS AND FURTHER DEVELOPMENTS

The model's ability to simulate the comprehension processes for the three tasks considered here is promising, though of course hardly conclusive. We need to expand the size of our long-term memory, the number of plan alternatives, and the range of problems considered, and see how the present model will perform. We need to explore more systematically than we have done so far the parameter values used in this simulation. (Basically, we chose the first combination of values that worked for the present simulation). We need to explore various ways of making our simulation more elegant and efficient (e.g. it may not be necessary to introduce a separate executive component to check whether plans are executable, etc.). Equally importantly, we need to find a way in which the details of the model predictions can be evaluated against empirical protocols. All we have done so far, is to note a generally satisfactory overall correspondence.

In spite of the fact that this work is as yet quite incomplete, our progress so far warrants some optimism. From the standpoint of problem solving work, what we have done is not spectacular: we do a means-end, backward problem solving that probably would not have challenged the General Problem Solver. However, we do this within a framework of a general theory of discourse comprehension. We are not proposing a model of problem solving in the computing domain *de novo* but merely adapt a comprehension model to this domain by representing the domain knowledge as plans. Should we be able to work out and test more fully the model sketched here, we plan to use the model to explore the question how novice users who do not have the kind of knowledge upon which behavior is based in our simulations can be helped to perform routine computing tasks.

REFERENCES

Card, S. K., Moran, T. P., & Newell, A. (1983). *The psychology of human-computer interaction*. Hillsdale, NJ: Erlbaum.

Kieras, D. E., and Polson, P. G. (1985). An approach to the formal analysis of user complexity. *International Journal of Man-Machine Studies*, **22**, 365-394.

Kintsch, W. (1985). Text processing: A psychological model. In T. A. van Dijk (Ed.), *Handbook of Discourse Analysis*, Vol. 2 (pp. 231-244). London: Academic Press.

Kintsch, W. (1988). The use of knowledge in discourse processing: A construction-integration model. *Psychological Review*, in press.

Kintsch, W. & van Dijk, T. A. (1978). Toward a model of text comprehension and production. *Psychological Review*, **85**, 363–394.

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

Kintsch, W. & Mannes, S. M. (1987). Generating scripts from memory. In E. vanderMeer & J. Hoffmann (Eds.), *Knowledge aided information processing* (pp. 61-80). Amsterdam: North-Holland.

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

van Dijk, T. A. & Kintsch, W. (1983). *Strategies of discourse comprehension*. New York: Academic Press.