The Application of Connectionism to Query Planning/Scheduling in Intelligent User Interfaces

Nicholas Short, Jr. *

Lokendra Shastri *

Abstract- In the mid nineties, The Earth Observing System will generate an estimated 10 Terabytes of data per day. This enormous amount of data will require the use of sophisticated technologies from real-time distributed Artificial Intelligence (AI) and data management. Without regard to the overall problems in distributed AI, this paper focuses on developing efficient models for doing query planning/scheduling in intelligent user interfaces that reside in a network environment. Before intelligent query/planning can be done, a model for real-time AI planning/scheduling must be developed. As Connectionist models (CM) have shown promise in increasing run-times, this paper proposes a connectionist approach to AI planning/scheduling. The solution involves merging a CM rule-based system to a general spreading activation model for the generation and selection of plans. The system was implemented in the Rochester Connectionist Simulator and runs on a Sun 3/260.

INTRODUCTION

The major mission of the National Space Science and Data Center (NSSDC) has been to archive and provide access to a wide variety of data from NASA's scientific experiments in the Earth and space disciplines. Historically, the NSSDC has been a centralized organization where all of the 2,500 online accesses per year for the over 125,000 tapes are sent to the Goddard facility (Green, 1989). While the volume of data requests can be currently handled, the arrival of many of NASA's new projects promises to cause not only a glut of data but also a large increase in number of data requests. This enormous amount will create a bottleneck at the centralized facility.

While an increase in the NSSDC's resources will assuage some of these problems, the authors believe that the size and complexity of the upcoming projects will require a distributed systems approach. For example, the NSSDC's data holdings will double every two years, reaching about 30 Terabytes by 1995 (Green, 1989). Moreover, projects like the Earth Observing System (EOS) will generate an estimated 10 Terabytes of data per day (Campbell, 1988). Considering that the NSSDC's current archive contains around 6 Terabytes (Green, 1989), it is doubtful that the current or future facility's resources can support the volume of requests

* Nicholas Short, Jr. is with NASA/Goddard Space Flight Center, Code 634, National Science Data Center, Greenbelt, Maryland 20771. He is also a graduate student at the University of Pennsylvania, Philadelphia PA 19104. Lokendra Shastri is a faculty member at the University of Pennsylvania.

without an introduction of new real-time distributed data management into the organizational structure.

Foreseeing this need, the NSSDC established the Intelligent Data Management (IDM) project, which has, as one of its goals, to develop advanced workstation tools for operation in a network environment. One component of this involves the use of intelligent user interfaces in a distributed architecture (Short, 1988). These interfaces will contain knowledge about the available resources and the access procedures not only from the NSSDC but from other archives as well.

One way to reduce the number of requests would be to guarantee that the queries to the facility are both accurate and supported by existing data. That is, if tools can be provided to the user that will allow him to develop intelligent queries at his own facility, then fewer ill-formed queries will be sent to the data center.

As is well known to researchers in real-time systems and database management, a critical ability required will be the need to plan and coordinate the various data access and manipulation tasks that are required to support a particular user's data goal. For example, suppose a user requires data sets A,B,C, and D and archives 1,2, and 3 contain some subset of the needed sets. Specifically, suppose

1) archive 1 contains A and B;

2) archive 2 contains A, B, and C;

3) archive 3 contains C and D.

Now, the obvious access plan would be to query either archive 2 and archive 3 or archive 1 and archive 3. The choice between these options will be determined from constraints such as network loads, the request arrival rates to the archives, processor loads at a particular archive, etc. All these choices must be resolved in real-time in spite of large number of expected data requests.

This problem of solving tasks based on constraints has been addressed by the planning sub-field of Artificial Intelligence, as well as the real-time systems field (Stakovic, 88). It would be apparent that solving the real-time constraints in distributed query planning will require a solution to the planning/scheduling problem.

Unfortunately, intelligent real-time planning in the AI field is only now beginning to be addressed by researchers. In this paper, we will explore the limitations of classical AI planning, propose a solution, and discuss some of the applications to the data management field. We hope that a minimization of requests to an archive can occur through the intelligent formulation and execution of queries through planning/scheduling.

QUERY PLANNING/SCHEDULING AND REAL-TIME SYSTEMS

Not surprisingly, scheduling of tasks for real-time systems cannot be based on algorithms which just try to satisfy deadlines. Stankovic states that

"The goal is to find optimal static schedules that minimize the response time for a given task set....The system is often highly dynamic, requiring on-line, adaptive scheduling algorithms" (Stankovic, 1988).

In fact, as these algorithms are NP-hard, the solution must be heuristic, forcing the problem into the AI domain (Zhao, 1987).

Any hope of solving this scheduling problem along with planning may have been thwarted by Chapman's proof that efficient, general-purpose planning is undecidable (Chapman, 1987). Yet, humans can solve the problem by, as Chapman states, "...improvisation, doing something easy and debugging the result when it fails " (Chapman, 1987).

This suggests that a solution may be to store numerous fixed plans, determine which plans are applicable to a given situation, and execute these plans efficiently until an error in execution occurs. Many trials (i.e., plan attempts) will be executed before the correct final plan is determined. That is, incremental planning must be done.

The incessant need to plan/replan will cause havoc for the scheduling algorithms which are trying to minimize response time for the systems. This delicate balance between response times and planning can be maintained by utilizing the most efficient method for heuristic planning/scheduling.

Connectionist Models (CM) have shown time reductions in several classical Artificial Intelligence (AI) algorithms (Shastri, 1989), but few CMs have been applied to the planning field (Whithead, 1989; Blelloch, 1986). In this paper, we present an initial attempt to move some of the functions of a classical planner into a CM. The idea involves merging spreading activation over a semantic task-net and rule-based inference into a CM to aid in the selection and generation of plans (Hendler, 1988; Shastri, 1989).

From the network system's perspective, The CM planner will reside on each node and will exist in the node's intelligent user interface. The user interface's knowledge-base will contain information about interactive problem solving among other network nodes. While this distributed AI problem is beyond this paper's scope, we will focus on the model of a node's CM planner without regard to specific details about cooperative planning over the network.

The role of the CM in a particular node's data management system will be to interface the high-level, symbolic components to the standard database models, as described in (Short, 1988). More specifically, expert database advisors, natural language front-ends, and graphics interfaces will comprise the high-level while database management systems will form the low-level.

PLANNING DESCRIPTION

In general, planning can be broken into two distinct phases: plan generation and planning decisions (Charniak, 1985). Plan generation resembles a deductive problem in that, given a task (goal) and a situation (database), we achieve that goal by solving any number of subtasks (subgoals). The product of plan generation is an "and/or" graph where the "and" branches correspond to ordered steps in a plan and the "or" branches match alternative plans.

Planning decisions, on the other hand, consist of two phases: plan coordination and plan selection. In the former, the partial-order produced by the plan generator is converted to a total-order. This is done by detecting plan failures that occur with some orderings of the "and/or" graph. In the latter, plan selection involves searching through a plan library to find the best alternatives (i.e., for the "or" branches). In this phase, the proper selection of plans can aid in the reduction of the "and/or" graph and, hence, the run-time of the planner.

In fact, if it can be shown that, while the plan generator is executing, a particular choice of plan can never be used given the current choice of plans, then no further reduction of that plan is required. For example, suppose that our planner is flown out to California on a trip and that while in California, it decides to buy a gun. If the planner has some prescience about airport security, it will realize that a return flight is impossible, because carrying a gun through a metal detector is illegal. There will be no need to reduce the subgoal "fly home" and an alternative plan like "drive home" could then be selected for reduction.

Analogously, some choices of plans could be chosen over others based on the current situation. For instance, suppose our planner wanted to determine the amount of deciduous forestation in Maryland and it contains an entry in a database for such information. In this situation, it would be better for the planner to select the alternative "database-query" as opposed to "calculate-from-raw-image-data." Choosing the former would save the planner from having to find the Landsat data, find an efficient processing environment (e.g., the MPP), determine the correct algorithms, transfer the data to the MPP, etc.

The output from the planner is a the total-order of tasks to be executed in order to achieve the goal. Each step in the total-order is a primitive task that corresponds to a leaf in the "and/or" graph and is defined a priori. One phase not included in the above discussion is the use of execution monitoring to provide feedback to the planner about its choice of plans. This module will notify the planner when error recovery routines must be invoked. Moreover, it can provide statistical information about outcomes in order to aid the plan selector.

RULE-BASED CONNECTIONISM

One of the first attempts to reduce the the run-times of planners was Hendler's Scraps model (Hendler, 1988). Scraps kept the plan generation and coordination modules in a symbolic logic-based system (i.e., NASL), while the plan selector was moved to a CM-like method using spreading activation over a task-network.

Although spreading activation was introduced by psychologists (i.e., by M. Quillian) as a cognitive model, it has proven useful to AI in reducing the costly time of unification of the rules to semantic networks. In general, the spreading activation is a bi-directional, breadth-first search over a semantic net. Beginning at two nodes, two searches are conducted by having each search mark its neighboring nodes as visited until the two searches intersect via a path. Once a path is found, it is returned to the logic system for unification against the rule-base. If unification fails, then more paths can be examined, as the spreading activation algorithm will continue independently of the logic system. Spreading activation in these terms can be viewed as a fast-subsetting mechanism of the semantic net. Of particular note, the nature of spreading activation makes it very amenable to implementation on parallel machines that act in background to the logic system.

In Hendler's model, a variant of spreading activation, called marker passing, is used where instead of just marking nodes as visited, nodes receive and save complex messages as marks. When coupled with the plan generator, the marker passer looks through memory to determine which plans the plan generator should reduce. That is, the marker passer would return paths to a plan evaluator which would either rule in or rule out choices. Specifically, the plan evaluator consisted of a set of heuristics which would reject or accept paths a viable before notifying the plan generator about plan choices.

The problem with Scraps, as with all marker passing systems, is that too many irrelevant paths can be returned. For example, in Charniak's WIMP system, only two paths out of 40 returned paths were usable from a semantic net of 75 nodes and 255 facts (Charniak, 1986). Hence, the path evaluator represented a bottle-neck that could make the time savings negligible.

One solution to the problem would be to move the plan generator into a CM in the hopes of getting rid of the plan evaluator. Unfortunately, attempts at moving more functionality into the planner have been limited due to what is commonly referred to as the "variable binding problem." For example, Hendler states

"Given a plan for 'MOVING X TO Y,' his (Blelloch, 1986) system must build special network components for each possible move that could be made. The range of X and Y must be predefined and the system can only plan on those. For example, given N blocks we generate the 2(1 + 2 + ...N)(N -1) plans for MOVE-A-TO-B, MOVE-A-TO-C, etc. These are then the only operators usable, and new blocks cannot be added without changing the system" (Hendler, 1988).

Another solution comes from the recognition that plan generation resembles logical inference. That is, if we can modify a reasonable CM deductive-retriever, then both the plan generator and marker passer could execute simultaneously in the CM. In fact, this paper presents a limited step at realizing this goal.

Basically, the approach is to modify the rule-based inference of (Shastri, 1989) to work with a spreading activation algorithm. In general, this is done by organizing the plans and goals into rules and the constants into a semantic net. Solutions to goals are then done by simultaneously proving a goal using the Shastri & Ajjanagadde model and spreading the activation over the constant semantic net. When a path is found that could rule-out a subgoal or plan, an inhibitory activation is sent over to the rule-based side of the network. That is, if a goal or task is ruled-out and the rule-based portion of the net has not tried proving it, then the inhibition will prevent the rule-based side from working on that subgoal.

Unfortunately, there can occur situations in which the rule-based side has already started proving the subgoal that was ruled-out. In this case, the inhibitory activation will send notification of its inhibition to the parent goals recursively up the "and/or" graph. Thus, both the rule-based model and the marker passer are in a race-condition. From the point of view of the planner, this presents no problem. However, the goal of reducing some of the time spent in plan generation may not have been met.

THE RULE-BASED CONNECTIONIST MODEL

In this connectionist model, there exists several types of units which correspond to predicates, arguments, etc. For each unit, several types of sites cluster the input

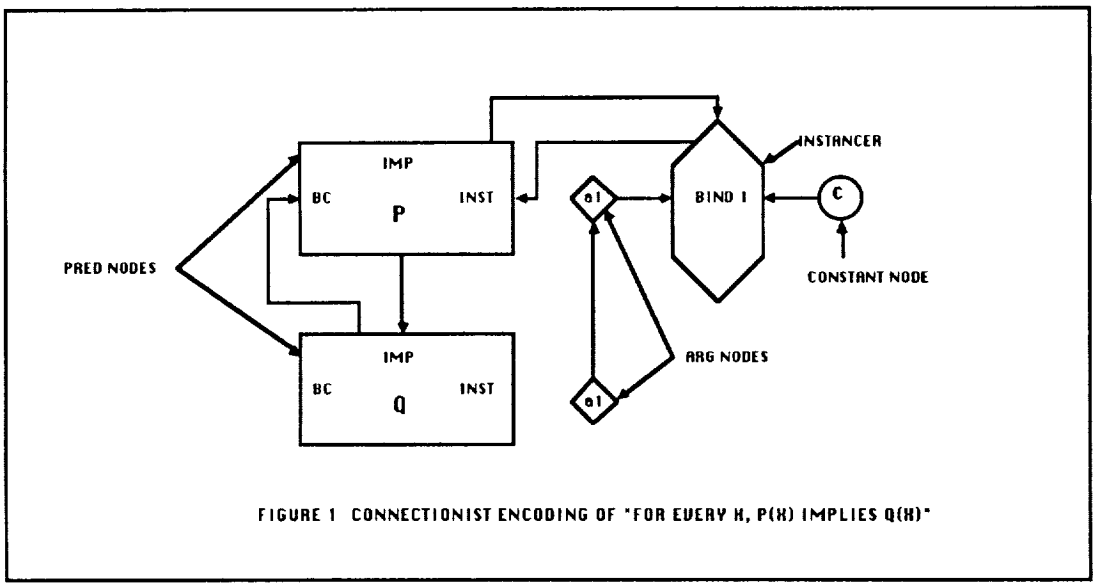

FIGURE 1 CONNECTIONIST ENCODING OF "FOR EVERY H, P(H) IMPLIES Q(H)"

connections from other units and modulate the inputs depending on the site characteristics. Connections between units are made to simulate both the database of assertions and the implications of the rules. Below, we provide an abstract description of the representation and reasoning in the rule-based system. For a complete discussion on the expressiveness of the rule-based system, see (Shastri, 1989; Shastri 1990).

In fig. 1, predicates are represented as rectangles, the associated arguments are represented as diamonds, and constants are represented as circles. For every predicate, there exists a number of associated argument nodes for every variable slot. The hexagons, called instancers, represent instantiated predicates. Suppose we wish to represent the rule:

(Every (x) (P x) => (Q x)) (*)

Also, suppose that we also know (P c) for some constant, c. If we wished to determine whether (Q c) were true, then we would activate the node representing Q, the argument node corresponding to Q, and the constant node c, allowing the network to run until (Q c) is proven true or false.

Now, each of these nodes is based on a node type called a binary threshold unit (BTU). That is,a BTU will output a 1 if any of its input values equals one. Otherwise, it will output 0. The key to controlling the spreading of activation is to make the activation of these BTUs phase sensitive.

The phase interval structure is defined by the query structure. In other words, the spreading activation is controlled by a clock where each cycle is broken into a fixed number of phases. The number of phases is dependent upon the number of bound arguments a the query. For example, one phase would be required to prove (R c) or (Love c y), for some constant, c, and variable, y.

A node type is partially characterized by the phase for which it can be activated. So, a constant node is active in the phase for which it was initially activated. For example, if the query (loves John Mary) were posed, the John and Mary constants

would always be active in the first and second phase, respectively. Similarly, an argument node becomes active in a phase i if it receives input from phase i in the previous cycle.

Predicate and instancer nodes are more complicated and are abstractions of BTUs. First, a predicate (pred) node contains three sites, IMP, INST, and BC, which collect all the connections from other nodes. Instead of just two states as in the BTU, the pred node has three internal and output states. The internal states are Inert, Enabled, and Active and the corresponding output states are 0, low, and high. The pred node changes state from Inert to Enabled if its BC site receives low or high input and from Enabled to Active if its INST site receives at least a low input or its IMP site receives a high input.

Second, an instancer node contains an Enable site and n bind sites that correspond to the arg nodes in the instantiated predicate. The Enable site receives input from the instantiated predicate's output link, while each of the bind sites receive input from both the arg nodes of the instantiated predicate and the corresponding constant. The instancer node becomes active at the end of a cycle if every bind site receives input from both the corresponding argument node and constant node. If only either the argument node or the constant node sends activation to the same site, then the instancer cannot be activated. Once active, the instancer node sends its output to the INST site of its pred node.

Rules are encoded by making connections among the aforementioned node types. That is, implication is enforced by making links between the pred nodes and arg nodes of the antecedent and consequences of the rule. If a variable in the consequence occurs in the antecedent, then a link is made from the consequence's arg node to the antecedent's arg node. Also, there will exist a link from the consequence's pred output to the antecedent's BC site and a link from the antecedent's pred output to the IMP site of the consequence node. So, for the rule (P x y) => (Q x), Q will have one arg node which connects to the first arg node of P's two arg nodes.

Then, when all the rules of the knowledge base are compiled into this formalism, the corresponding network forms a type of directed acyclic graph (DAG). The leaves of this DAG correspond to those antecedents which correspond to either asserted facts or antecedents which require no proof. Answering a query then corresponds to sending activation from the query's pred node backwards on the DAG until the leaves are reached. Once the leaves become active, activation is sent back along the IMP links to the original query's pred node. If the query pred node receives activation back, represented via the state of the pred node, then the query is considered true, otherwise it is considered false. Notice that the complexity of the proof is then twice the length of the longest path in the DAG.

For an example, suppose we wanted to prove (Q c) from (*). Because there is just one bound constant,c, in the query, there would be just one phase per cycle. In phase 1 of cycle 1, the Q pred node's state would be set to Enable and the arg node and constant node c would be set to 1. At phase 1 of cycle 2, P's pred node will be enabled by the link from Q's output and P's arg node will be enabled by the link from Q's arg node. It will then send output to the Enable site of the instancer node corresponding to the fact (P c). At this point, since both P's arg node and the c node are sending output to the instancer node, the instancer node is activated and sends output to back to the P pred node. This causes the P node to go from a state of Enable to Active. When this occurs, high activation is sent to the IMP site of Q. This causes Q to go from a state of Enable to Active suggesting that the proof worked. Had we not

had the fact that (P c) exists, the proof would have failed in the second cycle because the Bind site of the instancer node would not have allowed P to activate. That is, its Bind site would have been receiving activation from the arg node only.

SPREADING ACTIVATION SUBSECTION

THE SPREADING ACTIVATOR

Before getting into the details of the interaction between the spreading activator and the rule-based CM approach, the implementation of the spreading activator in the CM simulator will be described.

Generally, the implementation is a simplification of Hendler's marker passer. In Hendler's model complex markers consisting of fields like origin, fromnode, formula, zorch, etc. are passed from node to node. Because these markers violate the CM assumption that simple messages are passed, only one field, zorch, is used in the communication. Zorch is an attenuation mechanism that dampens the spreading of activation through the net. That is, as each mark is passed, the zorch factor is decreased by dividing it by the degree at each node. When zorch falls below a certain threshold, marking is stopped.

To start the spreading activation, the constants from the bound arguments in the initial goal are marked initially. In terms of the simulator, the unit's state is set to a value MARK and the potential and output are set to the initial zorch. To decrease zorch, the output links are weighted with the degree of the node. Zorch is then reduced at the site "MP" by dividing the link value by the weight (i.e., neighbor_num * 1000). If two zorch factors enter the site at the same time, then the smaller of the two is chosen.

In addition to reducing zorch, the site function at "MP" flips the pointer back to the originator of the activation. This is done so that the original path can be recovered when an intersection is found. Using this method avoids the problem of looping that Hendler's algorithm had to solve.

PATH EVALUATION

As aforementioned, Hendler's algorithm uses a set of heuristics to reduce the number of paths. When a path is returned from the marker passer, each heuristic examines the path to see if it is relevant to planning. Specifically, the five heuristics used are:

1) **QUICK REJECTION**: reject the paths that have already been examined,

2) **DEALING WITH DEMONS**: execute a demon when it occurs in the path,

3) **DEALING WITH PERCEPTUAL FLAGS**: "rule in" any tasks on a path that contains a perceptual flag,

4) **DEALING WITH FAIL FLAGS**: "rule out" any tasks on a path that contains a fail flag,

5) **DEALING WITH PLAN INTERACTIONS**: rule-in/rule-out plans that affect other choices in plan generation (e.g., the California plane trip example).

Of particular note, demons are rules that recognize certain conditions and interrupt the planner in order to add/modify plan steps. They are often used when a particular event must occur now, instead of later in the plan execution phase.

In addition to demons, flags are notes that are asserted into the semantic net when an important property has occurred. For example, when the planner is holding a gun, a flag is asserted into the net by the forward chaining rule

(-> (POSSESS ?x ?y) (FLAG ?y PERCEPTUAL (POSSES ?x ?y)

The assertion (FLAG 'gun PERCEPTUAL (POSSESS 'planner 'gun)) could be used by the path evaluator to rule-in the "shoot oneself" plan. Similarly, a fail flag like (FLAG FAIL (ON-STRIKE (CHEF ?x))) could rule-out a plan for a chef to cook a meal.

In our system, heuristic 1,3, and 4 can be replaced by the scheme discussed below. Although 2 and 5 may be possible in this scheme, they were not addressed.

First, heuristic 1 is implemented by a combination of the back links and spreading activation back from an intersection node. In detail, when two paths intersect, the intersecting node begins sending information back to the starting nodes. As this activation crosses the marked nodes, it checks the node-type, defined by the set membership in the simulator, to determine if activation should be sent to the rule-based portion. Since the rule-based nodes need only be excited or inhibited once to rule-in or rule-out a path, an activation crossing the marked nodes for a second time will have no effect on the corresponding rule-based node. Hence, duplicate paths are irrelevant.

The type-checking is actually implemented by a connecting link from the marked node to the corresponding node in the rule-based section. Details of how the rule-based section behaves will be discussed in the next section.

Second, heuristic 3 and 4 are implemented in a similar manner. Like Hendler's algorithm, flags are asserted into the net. However, when the marker passer crosses a flag, it checks the node type. If the node type is a fail flag, the zorch is negated and passed to its neighbors. This is done so that when an intersection occurs, the system knows that a rule-out should occur. That is, at an intersecting node, instead of sending a positive activation back to the origins, a negative or inhibitive value is returned. This tells the various nodes along the return path whether to excite or inhibit the corresponding rule-based nodes. Since this system only "rules in" or "rules out" plans, a positive activation implies a "rule in", whereas a negative implies a "rule-out."

RULE-BASED PLAN GENERATOR

Unfortunately, the CM rule-based implementation had to be modified to handle the interaction between it and the spreading activator. This amounted to changing many of the unit functions, site functions and behavior of the constant nodes.

First, in order for the variable bindings to work in the rule-based section, they must be activated in the phase corresponding to their position in the query. This will, however, disrupt the spreading activation as these constant nodes are participating in both plan generation and spreading activation over the constant task-net. That is, if a constant node activates in its phase it will not only send

activation to an instancer node but also another neighboring constant. The neighboring constant will mistake that activation as zorch.

The solution to the problem is to notice that the binding of constants occurs only in the first cycle. So, if spreading activation is delayed until the second cycle, the first cycle can be used to notify the instancer node about the phase in which they should be activated. In other words, another type of instancer node is created. This node records the phase in which it receives activation in the first cycle. Acting independently from the constant nodes, it then activates in its respective phase for the rest of the cycles.

Second, in order to rule-in/rule-out plans, a link exists between the constants in the spreading activator and the rule-based section. That is, the constants consist of three types: constants, tasks, and flags. This semantic net is organized like a task-oriented hierarchy as in fig. 2. For each node of type "task" (henceforth called task constant) a link is made from it to the corresponding pred node in the rule-based section. So, for each plan/task/etc. two nodes are required instead of the one used in the original rule-based CM. So, when activation from a returned path crosses the task constant, the pred node receives activation from the task constant as to whether to activate or shut down.

IMPLEMENTATION DETAILS

The model was implemented in the Rochester Connectionist Simulator (RCS) on a Sun 3/260 workstation. The RCS was chosen for its portability to numerous machines including a parallel machine, implementations in both Suntools and Xwindows, and its graphics interface.

A lisp interface to the planner was written to allow access from various expert system shells, including the Automated Reasoning Tool and the Advice Taker/Inquirer (ATI) (Cromp, 1988). Eventually, the ATI will be used to enter in plans from an expert for execution in the connectionist planner. Both the ATI and the connectionist planner could reside on several nodes on the network, where each node will contain heuristic knowledge about network resources, network traffic, and access procedures.

EXAMPLE

Suppose that the intelligent user interface on machine A has determined that the user wishes to "get" a file which exists on another machine B. To illustrate how the spreading activator could stop plan generation, suppose that the other machine's "get" command has been disabled. Because of this, initiating the file transfer protocol would be useless and we would want the planner to avoid reducing the ftp command.

Specifically, the following plan library solves this problem:

```
(To-do (user-request ?dataset ?location)
        (DataAccessPlan ?dataset ?location))
(Plan (DataAccessPlan ?dataset ?location)
        (steps (check-net-node-status ?location on)
```

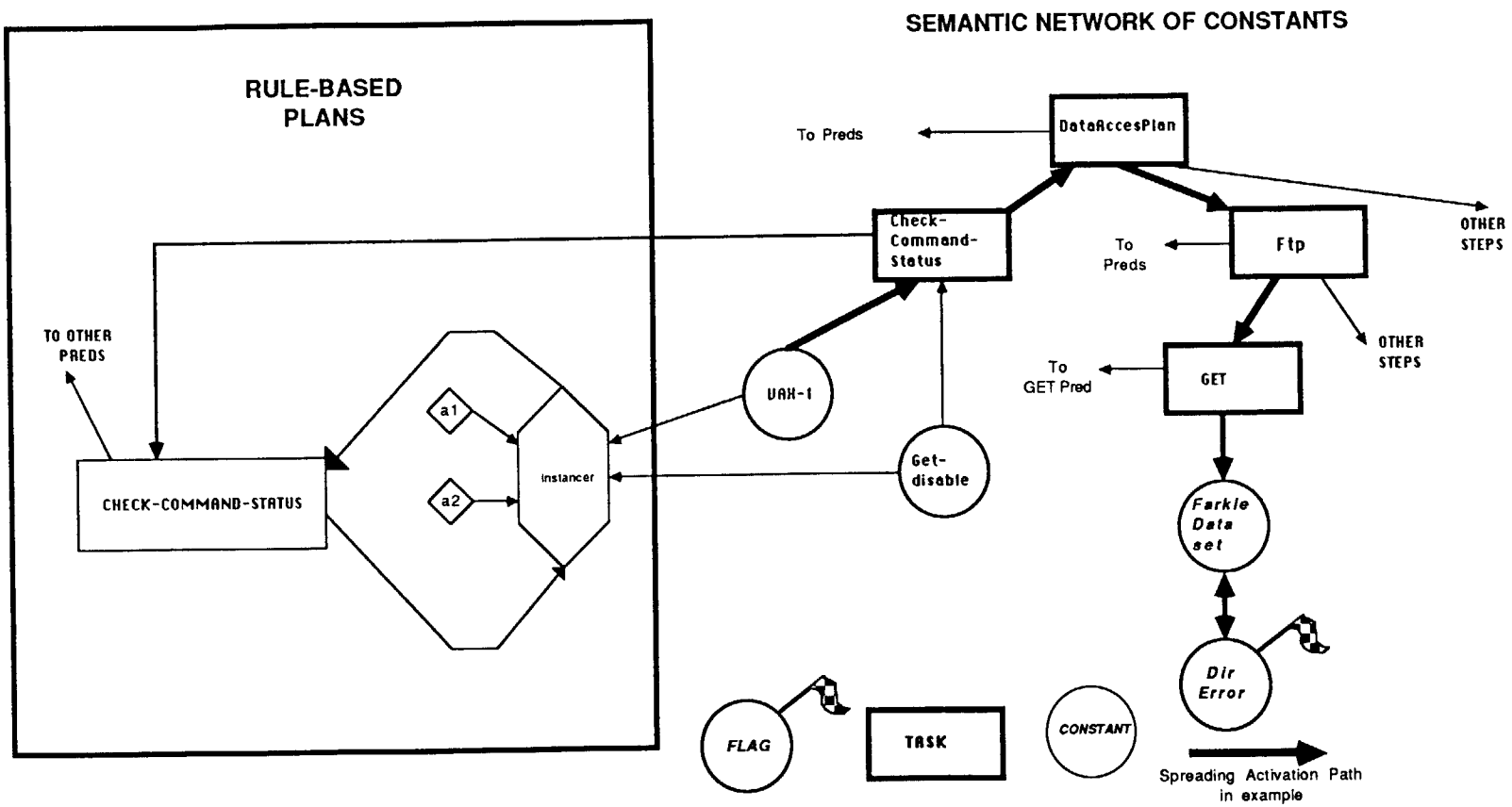

FIGURE 2. PLAN GENERATION AND SPREADING ACTIVATION ARENA

```
        (ftp ?dataset ?fromloc ?toloc)))

(Plan (ftp ?dataset ?floc ?toloc)

        (steps (open ?floc)

                (open ?toloc)

                (get ?dataset)

                (close ?floc)

                (close ?toloc)))
```

The connectionist implementation is shown in fig. 2 Because machine B's get is down, a flag is placed in the constant net. To solve this problem, the query

```
(solve '(user-request 'FarkleSet 'Vax-1))
```

is posed to the connectionist net. Because there are two arguments, we have two phases for every clock cycle. After the first clock cycle, the spreading activation is started in the semantic net on the Vax-1 and FarkleDataSet nodes. Eventually, a path will be found between the DirError node and Vax-1, causing negative, inhibitory values to be sent from the intersecting node back to the starting nodes. When the inhibitory scalars cross the Get node, inhibition is sent to the Get pred node on the rule-based side. This will shut off the Get pred, causing negative scalars to be sent back to parent pred nodes via the implication links.

Of particular note, many of the details for the choice of constant node in spreading activation have been left out. See (Hendler, 1988) for a better description.

CONCLUSION AND FUTURE DIRECTIONS

The advantages of moving the plan generator into rule-based CM are threefold. First, we have shown how to avoid part of the path evaluator in order to stop the bottle-neck between plan generator and plan selector. Second, we have reduced the expense of plan generation by going to the linear run-time of the rule-based CM. Lastly, we can still use our hierarchical representations while ignoring the implementation question.

Unfortunately, not all of Hendler's functionality was achieved. For instance, Rule-in in our model is somewhat meaningless in the CM. Because there is no way of knowing which argument nodes in the middle of the "and/or" graph will be activated by lower argument nodes, the CM planner must wait for the activation to come up. Rule-in's only use is to help the plan coordinator in choosing an alternative. One possible addition could be to have the ruled-in pred node laterally inhibit its sister alternatives. Then, when activation reaches the ruled-in node, values will propagate only through that node. Hence, no nodes are unnecessarily activated.

Much work still needs to be done to make this a viable model for planning. For example, a plan coordinator must be integrated naturally, as (Whithead, 1989) has suggested. Finally, if not all of the planner can be moved into a connectionist model, then the simple planning could be reserved for the CM, while complex planning

could be done in a classical symbolic planner. In other words, the planner first tries to plan and execute a plan sequence using the CM. If that fails, it then passes partial information up to the symbolic planner which uses that to generate a more complicated plan. This is, of course, exactly what Chapman suggested as a solution to the general planning problem.

While this research may not be significant in the "short-run" to NASA's data management problems, we argue that with NASA's appropriation of several parallel machines, connectionist models in general have the propensity to increase the efficiency for AI requirements in intelligent user interfaces. The major benefit of these models is that much of the classic AI algorithms (e.g., back-chaining) can be kept without any loss, as is not the case with more standard neural net approaches.

REFERENCES

Blelloch, G., (1986),"AFL-1: A Programming Language for Massively Concurrent Computers," MIT AI Laboratory, Technical Report AI-TR 918.

Charniak, E. (1986), and McDermott, D. ,*Introduction to Artificial Intelligence*, Addison-Wesley Publishers.

Charniak, E. (August, 1986), A Neat Theory of Marker Passing, *Proceedings from the Fifth National Conference on Artificial Intelligece* (pp. 584-88), Philadelphia, Pennsylvania.

Campbell, W., Short Jr., N. and Treinish, L.,(May, 1989) "Adding Intelligence to Scientific Data," *Computers In Physics*..

Chapman, D., (1987) "Planning for Conjunctive Goals," *Artificial Intelligence 32*..

Goddard, N., (1987) "The Rochester Connectionist Simulator: User Manual," Dept. of Computer Science, University of Rochester.

Cromp, R. (1988) "The Advice Taker/Inquirer, A System for High-Level Acquisition of Expert Knowledge, *Telematics and Informatics*, 5(3), pp. 297-312.

Green, J. (1989), "The New Space and Earth Science Informations Systems at NASA's Archive," Accepted for publication in *Government Information Quarterly.*

Hendler, J., (1988) *Integrating Marker-Passing and Problem Solving: A Spreading Activation Approach to Improved Choice in Planning*, Lawrence Erlbaum Associates, Publishers.

Shastri, L., (1988) *Semantic Networks: An Evidential Formalization and its Connectionist Realization,* Morgan Kaufmann Publishers, Inc.

Shastri, L. and Ajjanagadde V., (January1989) , "A Connectionist System for Rule Based Reasoning with Multi-Place Predicates and Variables," Computer and Information Science, MS-CIS-8905, Univ. of Pennsylvania.

Shastri, L. and Ajjanagadde V., (January1990) ,"From Simple Associations to Systematic Reasoning: A Connectionist Representation of Rules, Variables, and

Dynamic Bindings," Computer and Information Science, MS-CIS-90-05, Univ. of Pennsylvania.

Short, Jr., N. and Wattawa, S., (December 1988) "The Second Generation Intelligent User Interface for the Crustal Dynamics Data Information System," *Telematics and Informatics, 5(3)*, pp. 253-67.

Stankovic, J., (October 1988) "Misconceptions about Real-Time Computing," *IEEE Computer*, pgs. 10-19.

Whitehead, S. and Ballard, D., (1989) "Connectionist Designs on Planning," 1989 Summer Workshop on Connectionism, Carnegie-Mellon University.

Zhao, W., Ramamrithham, K., and Stankovic, J., (May 1987) "Scheduling Tasks with Resource Requirements in Hard Real-Time Systems," *IEEE Trans. Software Eng.*, Vol. SE-12, No. 5, pgs. 564-577.