Integrating Marker-Passing and Problem Solving

James A. Hendler
Department of Computer Science
Brown University[†]

In this paper we describe how an efficient underlying mechanism, a parallel, spreading activation algorithm, can be used during problem solving. We present this mechanism, chosen due to its demonstrated usefulness for several other cognitive tasks, and show how it can be used to guide the problem solver in choosing correct plans, rejecting plans that violate constraints, or modifying plans as they are generated. Examples of how this technique is used are given, and an implementation of such a system, integrating a marker-passer with a problem solver, is described. The paper discusses some of the desiderata in designing such systems and some of the issues that arise. Some future directions for the work are also described.

1. Introduction

A major problem faced by problem-solving systems today is that of making a choice in the presence of multiple alternatives. It is often the case that the system has access to knowledge that would lead to the optimal solution, or that could avoid a conflict, but this knowledge is not used. Consider the case of a system trying to solve a goal such as *Satisfy hunger*. It must make a choice between alternatives such as *Go to restaurant* and *Eat at home*. In the absence of other knowledge it may not matter which path is chosen, but what if we had already asserted *You have no money* to this system? At this point there is a conflict down the *restaurant* path, but none down the *eat at home* path. Thus the system should choose the latter. Unfortunately, most present day systems are unable to take advantage of this sort of information, and would make the choice at random. At a later point in the problem solving it would encounter the error (assuming it took the *restaurant* path) and be forced to backtrack. Avoiding backtracking has been a primary goal of problem-solving researchers.

In this paper we present a new approach to the issue of choosing paths during problem solving. We propose that a suitably structured "marker-passer", a parallel, spreading activation mechanism, can be used to aid the choice mechanism used by a problem-solver. In this paper we will describe an implementation of such a mechanism, as well as discussing why such a mechanism is desirable.

2. Integrating Marker-Passing and Problem Solving

Present day problem solvers work by using information found in Memory to generate plans. These plans are then subjected to some form of plan Evaluation, either in the form of demons (cf Sussman, 1975; Sacerdoti, 1977) or meta-rules (cf Wilensky, 1983) which critique the plans and if necessary, return them to the problem-solver for reworking.

Most problem solvers work by making step-wise refinements on subplans until primitive actions are reached. Thus *Restaurant* would be broken down into *enter, order, eat, pay,* and *leave. Order* could be broken down into *read menu, decide,* and *tell waitress*, etc. Steps such as *read menu* would be primitive and thus the final plan would be comprised of such steps.

It is this step-wise refinement that causes the problem described in the introduction. At the time we decide which plan to use we do not yet know what steps it will eventually decompose into. However, it is when we try to perform these primitives that the conflicts will most often manifest themselves.

[†] This research was supported by Office of Naval Research under contract N00014-79-C-0592

It is clear that if a mechanism could be developed that would examine all these primitive actions looking for possible conflicts and identifying them prior to the making of a choice then the problem solver would benefit. This mechanism would need to find all possible bindings for the variables being passed through the various levels of substeps checking for conflicts.

Unfortunately, such a mechanism does not presently exist. To be computationally viable this mechanism would need to perform deduction extremely quickly. Further, since all possible bindings of any variables need to be examined, large numbers (susceptible to combinatorial explosion) of low-level deductions[†] would need to be done. Parallelism would improve the situation, but not solve it, due to inherent limitations on the efficiency of such deductions and the combinatorics of multiple possibilities of variable bindings.

It is possible, however, to approximate this mechanism with a system that can examine the primitive actions quickly while looking for special features. A Marker-passing system is one such mechanism. It is our contention that by merging a marker-passer with a problem solver we can provide substantial improvement. Our system (Figure 1) has the marker-passer integrated with the problem-solver and plan evaluator. At this point we will discuss exactly what we mean by marker-passing and why we prefer this type of system. Following that we will show some examples of how the marker-passer is integrated into the problem solver.

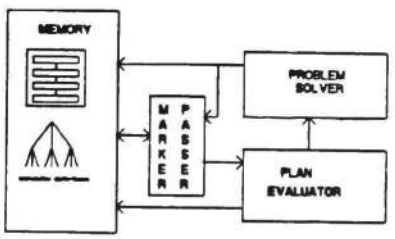

Figure 1: Adding a marker-passer to a Problem Solver.

2.1. What is Marker-Passing?

Marker-passing (cf Quillian, 1966; Fahlman, 1979; Charniak, 1983), a computational model of spreading activation, can be thought of as the marking of nodes adjacent to some node in memory, and then marking all nodes adjacent to those, etc. Thus, for example, in a traditional semantic net system if we marked *CAR* we would then mark *WHEELS*, *VEHICLE*, and all those concepts directly related to *CAR*. Following this we would then mark *RUBBER*, *ROUND* and all those concepts relating to *WHEEL*, as well as *TRANSPORTATION* and those concepts relating to *VEHICLE*. This process would proceed until either nothing was left to mark or the process was called to a halt.

By marking first one node and then another we can see how they are related by examining those concepts marked by both. Thus, in this example from Charniak (1983):

John wanted to commit suicide. He got a rope.

We would start passing markers at *suicide* and then pass markers starting at *rope*. This would find an intersection at the node *noose* which would be found both from *suicide* (the instrument of a *hang*) and from *rope* (as a material relation).

The primary difference between the marker-passer defined by Charniak (1983; based on Quillian, 1966) and the parallel model proposed by Fahlman (1979) is that the former will return the path found, rather than reporting the node of intersection (as would the latter). Thus in the example above the marker-passer would return:

SUICIDE -> to do suicide KILL self -> KILL -> HANG is a type of KILL -> HANG -> instrument of HANG is NOOSE -> NOOSE -> NOOSE is made of ROPE -> ROPE

[†] Depending on the type of problem-solver being used these computations could involve unification, pattern-matching, script instantiation, etc. In this paper all of these types of tasks will be referred to as "low-level deductions" for want of a better term

Notice that the marker-passer does not perform any deductions during its run. If we had asserted that the agent and patient of a *hang* had to be different this path would still be found. This would not be a valid path since the definition of *suicide* requires that the agent and patient must be the same. It is this "blind" behaviour that enables us to implement marker-passing efficiently, at the expense of finding some false paths. The issues of evaluating false paths and avoiding their generation are discussed later in this paper (section 4.2).

2.2. Why Marker-Passing?

In considering an underlying mechanism to use for problem-solving it was our desire to find a system with several properties: it must be able to examine many possibilities efficiently, it must return information that the problem-solver can use for making choices, and it would be desirable to start from an existing system, being used for other cognitive problems. Marker-passing fits all three of these criteria.

Earlier, we discussed how the choice mechanism needs to be able to access the primitive actions of our problem-solver. As the marker-passer proceeds from a plan it marks the substeps, the substeps of these, etc. until it reaches the primitives. Since it does not do deduction as it marks it is able to reach these nodes efficiently. Further, existing implementations of marker-passing (Fahlman, 1979; Charniak, Gavin, Hendler, 1983) are formulated so as to be computed in parallel[†] thus gaining efficiency. In section 3 we will show examples of how the problem-solver can take advantage of the marker-passer. Thus, we satisfy our first two criteria.

Our system is not, by any means, the first to propose the use of marker-passing as an underlying mechanism. Quillian (1966) described a system, TLC, in which spreading activation was used to analyze sentence fragments such as "lawyer's client." As these phrases were analyzed the system would "learn" their meaning and thus be able to build to an understanding of more complex sentences. Collins and Loftus (1975) expanded some of Quillian's ideas, and described the psychological relevance of the spreading activation idea.

Fahlman (1979) described a system in which a marker-passing like scheme was used as the basis for a deductive system. In his system, NETL, links between nodes had properties that determine how marks pass over these links. By allowing this feature, many paths that a blind marker-passer would find are avoided. For example, if we wanted to encode the information that "Clyde was a *pink* elephant" we would have a regular link from *Clyde* to *elephant* and another link, called a cancellation link, from *Clyde* to the area where we record that *the color of an elephant is grey*. Fahlman described an implementation scheme for doing many standard deductive inferences using these sorts of links.

Charniak (1983) described the use of the marker-passer as a device for aiding in the process of story comprehension. The marker-passer is used to find possible paths through the frame knowledge database that relate two frames together. These paths enable the system to build up a model of the frames being instantiated. Thus, in the example of section 2.1, the path from *suicide* to *rope* is found by the marker-passer. It returns this information to a higher level mechanism which uses the information for instantiating the *hang* frame and for building a representation of the events in the story.

Hirst (1983) used Charniak's model of marker-passing for word sense disambiguation. Part of his "Polaroid words" system would pass markers between the senses of words being examined. Thus, in the case of the sentence:

The farmer bought the straw.

the marker-passer would find the path between *farmer* and the *hay* meaning of *straw* thus preferring this meaning over the *drinking straw* meaning for which no significant connection would be found. Further, his system explains the difficulties that would be found in disambiguating a sentence such as:

[†]Due to hardware limitations these systems have been implemented as "psuedo-parallel." That is, although running on serial machines they violate no constraints of locality or temporality that a massively parallel system would need.

The astronomer married the star.

Granger, Eiselt, and Holbrook (1984) developed a model of parsing based on the spreading activation model. Their program, ATLAST, models the integration of lexical, syntactic, and pragmatic processes during natural language comprehension. A spreading activation model of memory is used to propose paths that are then evaluated by a filtering mechanism. Those paths found to be significant are pursued by the parser.

3. Examples

The flow of control when we integrate the marker-passer with the problem-solver was shown in figure 1. In general the following occurs: To start, the problem-solver is invoked. In attempting to solve problems it passes markers throughout the knowledge base. When these markers encounter other markers an intersection is found. The paths meeting at this intersection are then returned to the plan evaluation mechanism. This mechanism checks this path for information which can be used to help guide the problem-solver. If no such information is found, the path is ignored.

The system checks to see if this path proposes a solution to an existing problem (see section 3.1 below), causes a conflict (sections 3.2 and 3.3), or suggests a modification to an existing plan (section 3.4). If one of these is identified the system takes appropriate action choosing, rejecting, or modifying the plan as necessary. In this section we will show examples of these situations and informally describe how the system would work. In later sections we will discuss the specifics of this implementation in more detail.

3.1. Example 1

The first example we will examine is one in which the marker-passer can help us identify a correct solution to a problem with many alternatives. In this example we consider the case of

Trying to commit suicide while holding a gun.

Figure 2 shows a simplified view of the knowledge base used in this example.

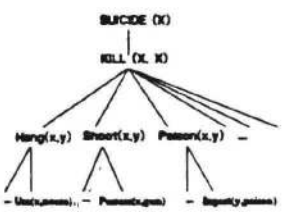

Figure 2.

The system needs to choose between each of the possible ways of commiting suicide shown. It starts by passing markers to each of the alternatives, these in turn mark their subplans, etc. While this goes on we are also passing markers starting with the frame[†] *gun*. Since this marks *gun*, and since *shoot* has as a precondition the possession of a *gun*, we find an intersection. The path

SUICIDE -> to do SUICIDE(x) KILL (x,x) -> KILL -> method of KILL(y,x) is SHOOT(y,x) ->
SHOOT -> precondition of SHOOT(y,x) is possess (y,GUN) -> GUN

is reported to the plan evaluator. Since this path presents a potential solution to the task which has been posed, committing suicide, the plan evaluator can cause the problem solver to prefer the *shoot* plan for achieving the *suicide* goal.

[†]We will use frames here as a generic word for a planning structure. The user may replace it with "script", "schema", "plan" or other such term.

3.2. Example 2

We can also use the marker-passer to reject plans that violate externally generated constraints. Consider, for example, the following situation:

> You are told at some time that the Air Traffic Controllers are on strike. At some later time you are told to plan a trip to California.

Figure 3 shows a simplified view of the knowledge we have in the database.

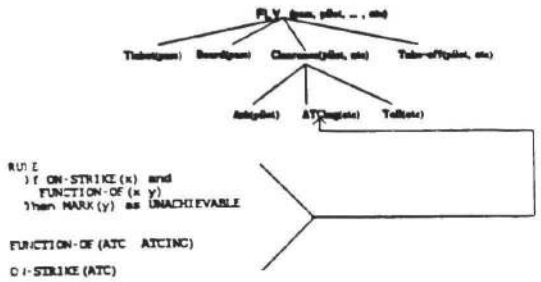

Figure 3.

It is important for our problem-solver to realize that the first statement (the ATC strike) is in conflict with our primary choice for the method of getting to California. When the system is given the first sentence it wishes to add it to the database. The system enters this information. It also notices a forward inference rule of the form *IF someone is on strike AND that person has some function THEN that function might not be able to happen.* Since the Air Traffic Controllers have as their function *ATCing* this would be marked.

We would now ask the system to plan a trip to California. Among the options to choose from will be the *FLY* frame. We will now pass markers on our alternatives. This will mark the *Ticket*, *Board*, *Clearance*, and *Take-off* frames, followed by each of the substeps of each of these frames. Since *ATCing* is one of the substeps of *clearance* it will be marked. At this point the marker-passer would recognise that an intersection has been found, and will return the path

> TRIP -> method trip FLY -> FLY -> step of FLY: CLEARANCE -> CLEARANCE -> step of CLEARANCE: ATCING -> ATCING -> ATCING (marked as unachievable by FACT of strike)

This information is passed to the plan evaluator which examines the path. Since ATCING has been marked as unachievable the system asserts that it cannot be done. The system now invokes a rule of the form *IF I cannot achieve a step in some plan, THEN I cannot achieve that plan.* Since ATCING is a step in the plan for FLY we can rule out the choice of flying as a method of transport for the problem solver.

3.3. Example 3

Another use of this mechanism is to identify conflicts between parts of plans. As an example consider the case of

> Buying a cleaver while on a business trip.

Figure 4 shows a simplified view of the information we use to process this example.

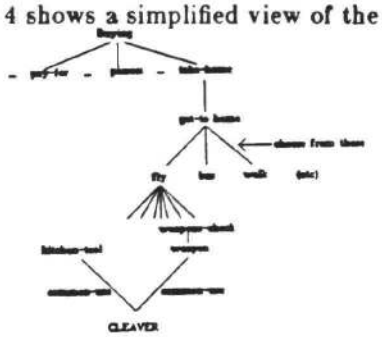

Figure 4.

The system starts by attempting to plan the buying of the cleaver. As such it passes markers down the paths for *Cleaver* and for *Buying*. From *cleaver* it use the knowledge that cleavers are used as *kitchen tools* and as *weapons* so it marks these nodes.

The system also marks the information about *buying* which includes the step of taking home that which is bought. Via the substeps of substeps the marker-passer eventually notices that one way to go home is to *fly* and that within *flying* one must pass through a *weapons check*. However, *weapons check* marks *weapon* and finds that it is already marked. Thus, an intersection has been reached and the information is passed to the plan evaluator.

The plan evaluator examines this path and discovers that the intersection involves carrying a weapon through a weapons check, which leads to being arrested, an undesirable state.

The plan evaluator can now use a heuristic which states that *IF a path causes one to reach an undesirable state THEN rule out the plan during which this state is enocuntered.* Since *Flying* is the plan which dominates the *weapons check* we can rule it out as the method of getting the cleaver home. It can then do replanning and either reject the purchase or amend the plan so as to avoid the illegality (and thus check the cleaver in as luggage).

3.4. Example 4

In the final example the marker-passer is used to find an inference rule that causes a modification of an existing plan. Take for example the following (from Wilensky, 1983):

The planner is given the task of fetching a newspaper from outside during a rainstorm.

Figure 5 shows the necessary information, simplified as usual.

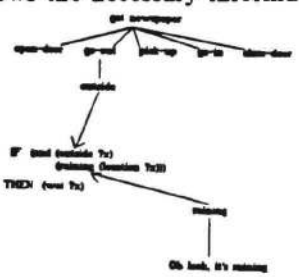

Figure 5

As before we pass markers from both parts of this problem. We thus mark the elements of the *get newspaper* plan and the *raining* statement. In this case we find the intersection not at one of the nodes of the plan, but at an inference rule in the system. This rule, *If it's raining AND if someone is outside THEN an umbrella should be used by that person*, is now examined by the plan evaluator. Since the person in question will can be bound with the agent of *get newspaper*, the plan will be modified so that that agent takes an umbrella.

4. SCRAPS: An Implementation

We have implemented a system, called SCRAPS, that combines a marker-passer and a problem-solver as described so far. In this section we will describe this implementation and discuss how we avoid some of the potential problems inherent in the design of such a system.

4.1. Implementation Details

SCRAPS is implemented using the problem solver NASL (McDermott, 1978; Charniak, 1982) on a logic-based semantics system, FRAIL (Charniak, Gavin, Hendler, 1983). Built into FRAIL is an implementation of a marker-passer. FRAIL and NASL are both implemented in the NISP dialect of Lisp (McDermott, 1983) and run on several different types of machines.

One important aspect of a marker-passing system has to do with the overall organization of the frame knowledge database being searched. For the purposes of performing marker passing, this database should be viewable as a large network of independent processors, each representing

a frame, that are interrelated via links along which markers may be passed. Each frame must be capable of both receiving and passing markers to neighboring frames through these links, which correspond to directed arcs relating frames to each other.

The FRAIL language provides such a system. Frames are translated into logical predicates which are stored in a LISP database as properties on atoms. These atoms serve as frames and the logical predicates serve as links. Thus, FRAIL provides both a logical language for deduction and a semantic-net representation useful for marker-passing.

The NASL problem solver uses FRAIL to retrieve the possible plans usable for satisfying a goal. When more than one plan can be applicable FRAIL uses a simple choice mechanism that checks for plans which are ruled out or ruled in. SCRAPS uses this mechanism as the primary communication between the plan evaluator and the problem-solver. When the marker-passer finds a path that the plan evaluator recognizes as a potential solution, that path is ruled in and all other paths are ruled out. Similarly, when the plan evaluator recognizes a problem it simply rules out that plan which would cause the contradiction. Synchronization is provided by having the problem-solver's choice mechanism wait until a message is received from the plan evaluator allowing it to proceed. Modifications to plans are done by having the plan evaluator make assertions directly into the FRAIL database.

As an example, consider once again the restaurant example of section 1. The system is started with the assertion that no money is available, and thus the *paying* frame is marked as unachievable via a forward chaining rule. We then ask the problem solver to solve the task of *eating* with some specific agent, *agent1*. NASL calls upon FRAIL to retrieve the potential plans and finds both *restaurant* and *eat at home*. As FRAIL retrieves these plans markers are passed. The path from *restaurant* to *money* is found and reported to the Plan Evaluator. The Plan Evaluator recognizes the conflict in the restaurant plan and therefore generates a *rule out* statement. [†] It then signals NASL to continue. NASL now uses its choice mechanism to examine the possibilities. Since the restaurant frame has been ruled out it uses the only other possibility, eat at home, and continues on.

4.2. Implementation Issues

For our system to be efficient it must be the case that the added efficiency of avoiding backtracking outweighs the overhead time spent in evaluating the returns from the marker-passer. If the plan evaluator takes longer to reject false paths than it would take the problem solver to find the correct path then the addition of a marker-passer loses its value. The plan evaluator must be designed so that it can quickly reject those paths with nothing to offer.

In the SCRAPS system the plan evaluator is composed of a set of *ad hoc* heuristics that can examine a path and see if any useful information can be gleaned. These rules serve two purposes: first, we wish to quickly reject paths which are not going to yield useful information, and second, we wish to be able to extract information from those paths which are useful. This is done by passing the path through a series of rules in a cascaded manner. The early rules are designed to reject paths known not to be useful, the later rules are designed to quickly check for a certain feature and pass the rule on to the next heuristic if it is not found.

The early rules work by first checking the node of intersection of a path. If the node of intersection is found to be a "promiscuous" node, one with a very high out-branching factor, the path is probably not useful and can therefore be rejected. If the node of intersection is not of this form, the path is checked to see if certain features can be detected that will cause the rule to be rejected. For example, a rule checks to see if a path is an *ISA Plateau*. This would be the situation if we passed markers from, for example, *dog* and *cat* and found an intersection at *animal*. These sorts of paths are useful for the word sense disambiguation use of marker-passing, but not for problem solving. Eugene Charniak (fortcoming) has been examining the paths returned by the marker-passer and trying to formalize path checking as a resolution proof procedure. As this

[†] Readers wanting more details about the format of the rules used by the choice mechanism in FRAIL and NASL are directed to Charniak, Gavin, and Hendler (1983).

work becomes practical our system will be changed to use this method for the rejecting of false paths.

Once the obviously useless paths have been rejected the Plan Evaluator then uses the second type of heuristic to see if the path yields valuable data. This is done by having each heuristic designed to quickly examine the path for some feature. If that feature is not found the path evaluator immediately passes this path on to the next heuristic. If none of these heuristics find such a feature then the path is considered uninteresting and rejected. If, however, such a feature is found the path is subjected to closer inspection to see if useful information is yielded. Due to the underlying logical formalism of FRAIL unification is used as a first test in these situations. If items in the path do not unify it can be rejected.

As an example of these heuristics consider what the Plan Evaluator would do with the *restaurant* example. With variables included, the path found is of the form:

> GOAL EAT (*me*) -> EAT -> To-do EAT(x) use RESTAURANT(x) -> RESTAURANT -> step-of RESTAURANT(x) is PAYING (x y) -> PAYING -> PAYING(*me*) is Marked as unachievable

We are also informed that the place where the marks intersect is on the node *Paying*. First we examine the node *Paying* to see if it is inherently uninteresting or "promiscuous." Since it isn't we go on to check the form of the path to see if we have found an ISA plateau (a very efficient heuristic). Again, this test fails, so we do not reject this path. We now move on to the second type of heuristic, those designed to extract information. The first heuristic to fire might be the heuristic checking for meta-rules of the type found in example 3.4. This rule would check the path to see if any of the links was of the form *Meta-rule*. Since none of the links in the path above have this form it is passed on to the next heuristic. This rule checks to see if either of the end-points of the path is marked as unachievable. Since this is the case, the rule is now subjected to closer scrutiny. The unification algorithm is used to see if variables in the path match up. Since, in this case, the person with the *Eat* goal is the same as the person trying to do the *Paying* the unifier reports the match. We now can use this rule to rule out using the restaurant plan as described in section 4.1.

Although, the key factor in designing these heuristics is to make the tests for path rejection as efficient as possible, it is still essential that the number of false paths generated can be held down. If too many paths are found by the marker-passer we lose efficiency since the evaluator must examine all those paths which are reported. If too few paths are reported the correct information can be missed. The marker-passer must be designed so as to constrain the number of paths reported.

We are presently exploring some of the issues involved in the design of such constraints. The marker-passer in FRAIL uses a propagation limitation on markers to help keep down the number of false paths found. Marker-passing is invoked with a certain strength at a starting node. That node divides the strength by the number of nodes it has as neighbors, and passes a marker to each of them with this new strength. Each of these nodes proceeds in turn marking its neighbors with ever decreasing strength. Once strength falls below a certain limit marker-passing stops along that path. Thus, long thin trails are preferred to short multiply branching ones.

Another means of limiting the number of false paths is the removal of marks during processing. At present we have developed a model which degrades marker strength over time, depending on the original strength of the marking. When the strength is sufficiently decremented the mark is removed. Recent work (Granger, Holbrook, 1983), however, has suggested that the removal of marks is a more complex process than the one we have implemented. We are presently examining this issue.

5. Research Directions

Recent work (Wilensky, 1983) has been examining the issues involved when a single agent has multiple goals, or when multiple agents, each with their own goals, are involved in a situation. This work has concentrated on recognizing, and resolving, conflicts between these goals. In section 3.3 we showed an example of how a system like SCRAPS could be used to detect problems

caused by a conflict between subgoals during a problem-solving task. Our recent research has been directed at exploring the relationship between these sorts of subgoal conflicts and the multiple agent goal conflicts being examined by Wilensky. It is our belief that SCRAPS can be extended to allow the plan evaluator to work on such tasks.

Take the following example (based on an example from Wilensky, 1978):

> Bill wants to get the treasure, but the hoard was guarded by a dragon.

SCRAPS would be invoked to solve two tasks one in which Bill attempts to obtain a treasure, and one in which a dragon attempts to guard a hoard. The marker-passer finds the path from Bill to the dragon via the path relating the treasure to the hoard. The plan evaluator can now examine this path. Since, in this simple case, the variables can be bound consistently the plan evaluator can recognize a potential planning conflict. If it had been the case that Bill had decided that the treasure he wanted was not the hoard but the giant pearl being guarded by the huge octopus, the plan evaluator would fail to match the dragon's treasure and the goal of Bill's quest, and thus the path would be regarded as false and therefore rejected.

The primary problem with extending SCRAPS to handle multiple agent planning is the growth in the number of false paths reported when common objects are used in many of the plans. Consider the following situation:

> John wishes to buy a farm. Mary, no relation to John, wants to get a job so she can earn more money. Bill, no relation to either of the others, wishes to go to a restaurant.

Our marker-passer will return paths connecting all of these as potential planning conflicts dealing with *money*. Similar growth in paths are caused by items like *cars*, *clothes*, and any other item common to many plans.

The single-agent version of SCRAPS is able to use a heuristic which rejects paths which meet at common objects. Thus, in the cleaver example of section 3.3 a false path such as:

> Cleaver -> ... -> weapons-check -> AGENT of Weapons-check is PERSON ->
> -> PERSON -> AGENT of buying is PERSON -> BUYING

can be rejected since the node of intersection, *person*, is involved in so many frames. In the multi-agent cases, however, these false paths cannot be rejected as simply. It now requires using our knowledge base to recognize that, for example, Bill's money, Mary's money, and John's money are all different.[†] We are presently working on redesigning the plan evaluator to handle such cases.

6. Conclusions

Integrating marker-passing with problem solving enables us to avoid backtracking in many problem solving tasks. An efficient underlying mechanism, the marker-passer, is used to choose correct plans, reject plans violating constraints, or to modify existing plans. The marker-passing mechanism is chosen due to its demonstrated usefulness during cognitive tasks.

We have shown that careful attention must be paid to the design of the marker-passer and the plan evaluator for the problem solving process to gain efficiency. At present SCRAPS is able to provide this efficiency boost in several situations. We are presently examining other problem solving areas to see if other classes of problem solving behaviours can benefit from this technique.

7. Acknowledgements

The author wishes to acknowledge Eugene Charniak for much aid in the design and critiquing of the work described in this report. Several of the examples used in this paper are based on his work. We also wish to acknowledge Doug Wong's aid in helping to formalize some of the pitfalls inherent in the design of a marker-passer.

[†] To convince oneself that this is true, consider the case where John and Mary are married and share a common bank account.

8. References

Charniak, E. *Micro-Nasl Reference Manual* Dept. of Computer Science, Brown University, 1982.

Charniak, E. "Passing markers: A theory of contextual influence in language comprehension." *Cognitive Science* 7(*3*), July - Sept. 1983.

Charniak, E. *A Neat Theory of Marker-Passing* Dept. of Computer Science, Brown University, forthcoming.

Charniak, E., Gavin, M.K. and Hendler, J.A. *The FRAIL/NASL Reference Manual.* Brown University Dept. of Computer Science Technical Report No. CS-83-06, Feb. 1983.

Collins, A.M. and Loftus, E.F. "A Spreading-activation theory of semantic processing" *Psychological Review* 82(*6*) pp. 407-428, 1975.

Fahlman, S.E. *NETL: A system for representing and using real-world knowledge* MIT Press, 1979.

Granger, R.H., Eiselt, K.P., and Holbrook, J.K. *Parsing with Parallelism: A Spreading Activation Model of Inference Processing During Text Understanding* University of California at Irvine Artificial Intelligence Project Technical Report #228, Sept. 1984.

Granger, R.H. and Holbrook, J.K. *Perseverers, Recencies, and Deferrers: New experimental Evidence for Multiple Inference Strategies in Understanding.* University of California at Irvine Artificial Intelligence Project Technical Report #195, May 1983.

Hirst. G.J. *Semantic Interpretation Against Ambiguity* Brown University Computer Science Technical Report CS-83-25, Dec. 1983.

McDermott, D.V. "Planning and acting" *Cognitive Science* **2** pp. 71-109, April 1978.

McDermott, D.V. *The NISP Manual* Yale University Computer Science Tech Report U/DCS/RR No. 274, June 1983.

Quillian, M.R. *Semantic Memory* (Scientific Report No. 2) Bolt, Beranek, and Newman, 1966.

Sacerdoti, E.C. *A Structure for Plans and Behavior* Elsevier, 1977.

Sussman, G.J. *A Computer Model of Skill Acquisition* Elsevier, 1975.

Wilensky, R. *Understanding Goal-Based Stories* Yale University Computer Science Research Report No. 140, Sept. 1978

Wilensky, R. *Planning and Understanding* Addison-Wesley, 1983.