Context Maintenance

Charles J. Petrie, Jr.
Microelectronics and Computer Technology Corporation
Advanced Computing Technology
Artificial Intelligence Laboratory
3500 West Balcones Center Drive
Austin, TX 78759

Abstract

Traditional applications of "Truth Maintenance Systems"(TMSs) fail to adequately represent heuristic search in which some paths are initially preferred. What they miss is the idea of switching contexts rationally based on heuristic preferences. We show that it is useful, especially for plans with contingencies, to maintain the validity of the reason for context choices and rejections. We demonstrate how to do so with a problem solver/TMS architecture called REDUX.

Scheduling Professors

Suppose we want to assign class sections to university professors for the coming year, and revise assignments as necessary. Taken in full generality, this is a formidable problem [17, 4]. Here, we extract a small fragment to illustrate four characteristics that distinguish a class of planning problems that are not well-supported by current techniques.

First, a characteristic of interest in such problems is that there are multiple objectives and constraints that cannot all be completely satisfied in a single schedule. Second, it is at least difficult to define an optimal schedule that trades-off ideally the various objectives, if not impossible, and certainly unnecessary. It is appropriate for this problem to compromise and find a single "good" schedule. Schedules that lack bad features are easier to identify than optimal ones. Third, although evaluating all possibilities is intractable, there exist heuristics for initial search and for resolving local conflicts that tend to result in good schedules. Fourth, contingencies force incremental revision of the solution.

One heuristic is to assign initially the courses to the professors that they wish to teach. A second heuristic is to assign courses to the more senior professors first. These heuristics can be justified in terms of constraint-directed search.[8] Our purpose here is not to derive such guiding heuristics but to provide a framework for using the ones available.

Suppose that professor Descartes wants to teach PHL305 in the Fall, so we assign him a section of that class, PHL305-1. A less senior professor, Kant, also wants to teach PHL305 in the Fall. But if we assign him a second section, PHL305-2, this violates a departmental constraint that there can only be one one PHL305 taught per semester. So, perhaps we reject this second assignment and instead give Kant a section of PHL380, which he indicated was his alternative choice to PHL305.

Expectations determine explanation requirements. Kant will probably ask "why didn't I get PHL305?". Answering this question is not just of practical use: this question points to a feature of problem solving important for revision: context rationale.

Context Rationale

We begin with a simple notion of "context", to be extended in Section , based on the standard, discrete, finite constraint satisfaction problem (CSP) [23]. A CSP has a set of variables $V = \{v_1, \ldots, v_n\}$, where each v_i has a domain $\{a_{i,1}, \ldots, a_{i,m}\}$ of possible values. Each k-ary constraint connecting variables $x_1, \ldots, x_k$ defines a set of k-tuples of allowable values for these variables. A k-ary constraint connecting variables $x_1, \ldots, x_k$ is defined as the set of allowed k-tuples for these k variables.

A solution set for a CSP is a set of tuples of values for all of the variables that are consistent with the allowed tuple sets for all constraint definitions. Domain-independent techniques for efficiently determining this solution set have been studied extensively [9]. Our course assignment problem requires only a single set element, and we have local heuristics that suggest both variable and value choices in a search for such a solution. The tentative[14] search technique of generate-and-test using variable ordering and dependency-directed backtracking is an appropriate technique for such a problem.

The set of choices of value assignments at any point in problem solving constitutes a *context* for further problem solving. Each new choice defines a new context. In Figure 1, variable V_1 was assigned the value a, then V_2 the value c, and V_3 has been chosen as the variable to assign next, but no value has been assigned yet. The problem context now is just the two assignments that have been made.[1]

[1]We want to keep track of the value assignments that have been made, since these are revisable. But once we

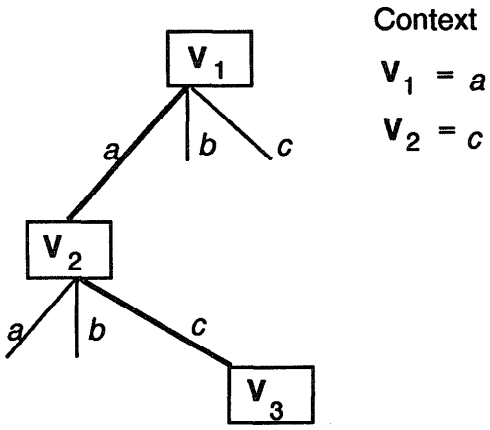

Figure 1: A Context

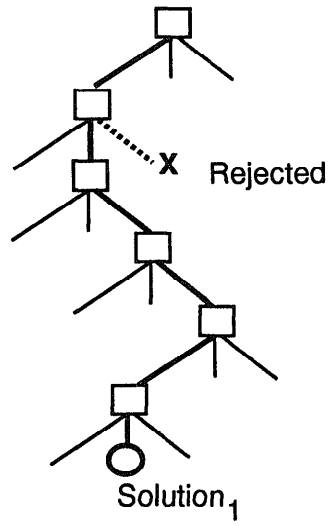

Figure 2: Path To A Solution

If the problem is nearly decomposable [24], attempting to make first assignments that maximally satisfy multiple objectives and testing for global constraint violations facilitates problem formulation (which is part of problem solving). Unlike MOLGEN[26], we pursue only a single solution and there is local heuristic guidance that gives us default paths through the variable assignment choices. If assigning a to V_1 corresponds to assigning PHL305 to Descartes, and assigning c to V_2 is assigning a section of PHL305 to Kant, then Figure 1 illustrates two default paths and a default context resulting from using the heuristic of assigning teacher preferences first.[2]

These default paths constitute a default context. When the defaults represent objectives, this is the unconstrained ideal solution that maximally achieves all objectives. Such default paths constitute *default reasoning* in design; we prefer to use cheaper or more reliable components and wait to see if that violates any constraints among the design choices. In any case, a default context has a *rationale*: the heuristic inferences for each choice.

Rejection of one or more choices may be required to resolve the violation of a constraint without relaxing it. In our example, assigning sections of PHL305 in the Fall to both Descartes and Kant violated a constraint. In this case, we switched from context $\{V_1 = a, V_2 = c\}$ to a less preferred one, say, $\{V_1 = a, V_2 = b\}$. The rationale for the new context includes the reason for rejection of the default.

Suppose such a context switch is made and the problem solver continues to a solution. The search path will have a form similar to that of Figure 2. There will be a solution context, which in this example includes $\{V_1 = a, V_2 = b\}$. There is also a rejected context consisting of $\{V_1 = a, V_2 = c\}$. The reason for rejection, and thus for the current context, may be nonmonotonic.

Suppose the assignment $V_1 = a$ is later involved in another constraint violation and we reject it. For instance, perhaps Descartes ends up with too heavy a teaching load and we must lighten it. We may take away his PHL305 course. The rejected assignment $V_2 = c$ is now consistent and at least locally preferred to the current $V_2 = b$.

If the problem solver does not detect the fact that changing the assignment of V_1 may allow a better assignment of V_2, then the rejection in the search path represents an unnecessary pruning of a better solution. We may be able to give Kant his requested course after all.

There must be a way to detect such unnecessary solution pruning so that opportunities to improve the solution are not missed. Detection of such possibilities is the computational importance of maintaining a rationale for the current context. The task of *context maintenance* is detecting current context rationale invalidity and notifying the problem solver of the possibility of improving the solution.[3]

It is well-known that recording bad combinations such as $\{V_1 = a, V_2 = c\}$ as *nogoods* is useful because we can then easily avoid generating it again. The nogood is necessary for the rationale for the context revision: why the default path was rejected and Kant did not get his requested section. The computational catch is that a nogood may be conditional.

When we design a plan, by making choices about

have made a choice about which variables to assign first, any performance loss is irreversible. So context only tracks choices, not order of choices.

[2]These heuristics may not be, and are not in this example, an evaluation function for a best-first search. Often such heuristics represent independent objectives for which no global metric is defined.

[3]Search heuristics may not be static but rather depend upon nonmonotonic reasoning. Context maintenance is thus required apart from context switches.

actions, we may add assumptions about the future state of the world as part of our default reasoning, and so introduce plan contingencies. Descartes may get a grant and not have to teach that semester at all. Or it may be that the constraint itself is negated by unexpectedly high enrollments. Kant may be able to teach PHL305 in both cases. Also, we may need to relax constraints when the problem is overconstrained.

What kind of computational support is available to track the rationale for the current (perhaps partial) solution context taking into account such possibilities?

Truth Maintenance

One AI hope for supporting such problems has been truth maintenance. The assignment of teachers in a course schedule is a plan that can be altered by unexpected changes in the world or by new plan decisions. Truth maintenance is potentially a general technique for precisely propagating the effects of such plan changes.

There are several varieties of truth maintenance systems (TMS). One major variety, the Assumption-based TMS (ATMS)[1], best supports problems in which one only wants to know what set of contexts are consistent with the constraints [2, 15]. For problems such as our course assignment example in which a single solution out of many is sought and in which nonmonotonic rationales are prevalent, we prefer the earlier TMS of Doyle[6] with the dependency-directed backtracking modifications of [15]. Some other systems that are fundamentally oriented for reasoning within a single context are [5, 10, 7]. We briefly review how a TMS usually works with a rule-based problem solver.

Suppose we represent our assignment choice heuristics by rules, such as "assign the preferred course for a teacher". Such a rule might look like:

```
Assign (?teacher ?course) if
 Prefers (?teacher ?course),
 Unless (Unavailable (?teacher)),
 Unless (Rejected-Assignment(?teacher ?course))
```

When the variables, denoted by ?, are bound at runtime, the instantiated left-hand side of the rule, the consequent, is added to the database and given a *justification* constructed from the instantiated rule antecedent. This justification is shown in Figure 3.

The arrow of the justification denotes the database node supported by the justification. The signed lines show supporting nodes. The *Unless* antecedents define the elements of the OUT-list of the justification, denoted by negatively signed lines. The single positive antecedent is the lone element of the IN-list of the justification and is denoted with the positively signed line. It has a *premise*[6] justification: its belief depends on no other node. The OUT-list elements have no justification.

The TMS has the task of labeling consistently the nodes in such a dependency network while avoiding certain circularities. For further details, see [15]. Here we note only that a justification is valid when each of its

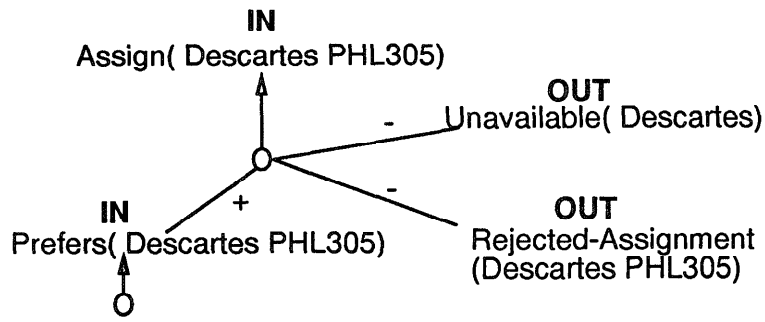

Figure 3: A TMS Justification

IN-list elements is labeled IN and each of its OUT-list elements is labeled OUT. The labeling is consistent if each node that has at least one valid justification is labeled IN and all others are labeled OUT. The labels must be updated when new nodes or justifications are added to the network. This syntactic relabeling is the TMS task.

In a TMS, a semantic conflict, such as a constraint violation, is represented by a *contradiction* node. The derivation of the contradiction determines its justification. In the example, the two assignments of Descartes and Kant would be in the IN-list of the justification for the contradiction. In dependency-directed backtracking, the dependency network is searched for the contributing assumptions: *culprits*[6] that have nonempty OUT-lists. If some *elective*[15] on such an OUT-list can be justified, then that justification for a culprit will be invalid. If the right justifications are made for the right culprits invalid, the contradiction/constraint violation is resolved.

In the example, we want to justify the rejection of the assignment of PHL305 to Kant. The elective justification should be *safe* and *complete* [15]. Completeness is the important feature here. It expresses the property that Kant's assignment should not be rejected unnecessarily. An elective justification is valid only as long as it needs to be. In particular, if Descartes' assignment of PHL305 goes OUT, the justification of Kant's rejection should become invalid.

Figure 4 shows the elective justification of the rejection of Kant's PHL305 assignment.[4] It simply depends upon Descartes' assignment to PHL305 and the nogood recording that the two assignments together are a bad combination not to be chosen again. The justifications of these are omitted. The one of the former has already been shown in Figure 3 and the latter is complex and discussed in [15]. Figure 4 also shows how the alternative choice of PHL380 might be justified for Kant using the rejection of PHL305. If the assignment of Descartes of PHL305 goes OUT, the assignment of PHL380 to

[4]This is a simplification of the elective justification of [15] which is a revision of the conditional proof justification of [6].

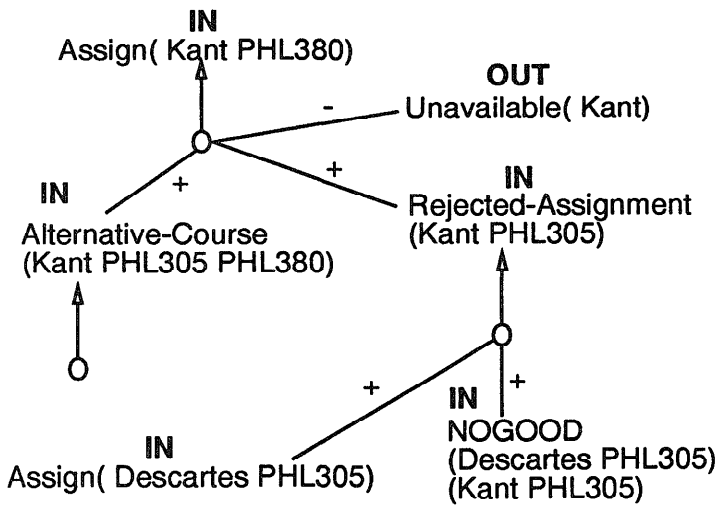

Figure 4: Alternative Assignment Justification

Kant goes OUT and the assignment of PHL305 to Kant comes back IN.

The completeness property of the elective justification is powerful. Any proposition in the dependency network that was necessary for the derivation of the constraint violation is linked to the elective justification such that if the derivation becomes invalid, through any means other than the elective being IN, then the justification will be invalid. If the constraint violation depended upon an assumption about the future state of the world, student enrollment for example, this is captured in the justification in the right way automatically. If actual enrollment violates this assumption, or if there is some other way in which the constraint ought to be relaxed, the justification becomes invalid.

What Doesn't Work

The potential of truth maintenance has scarcely been used in design, planning, and scheduling.[18] The traditional problem solver/TMS architecture outlined in Section has three key, related problems. First, it uses the mistaken principle that it does not matter what inferences the problem solver makes: the TMS caches all inferences alike and enforces consistency among them. Second, contradictions are included in the consistency the TMS is supposed to enforce. Third, there is no differentiation among the types of assumptions made during problems solving.

This mistaken principle of inferential indifference makes applications difficult to write and their behavior unpredictable. The primary source of difficulty is that there is no syntactic difference between inferences and metainferences about inferences; control inferences. Consider the agenda search strategy: "Choose the most important undone task, accomplish it, and mark it done." If this is achieved by a chain of rules in which all dependencies are recorded, it must result in a dependency network that the TMS cannot consistently label. The justification for the mark will depend upon the lack of the mark. Even if such an *unsatisfiable circularity* can be avoided by careful rule/dependency editing, the mark of completion may still depend upon nonmonotonic factors used to choose the most important task. But the heuristic of putting out the closest fire does not necessitate rekindling the fire if another fire subsequently is discovered to be closer.

If such control heuristics are distinguished, Section points out that some of these need not be recorded with a TMS; e.g., the choice of a variable or a goal on an agenda. Thus the unsatisfiable circularity problem can be avoided. But that same section points out that not recording context rationales means missing opportunities to improve solutions. And context rationales contain heuristic search guidance. The problem with including this kind of control information in the justification of the inference consequent, such as a value assignment, is that the TMS labeling may be too aggressive.

Suppose a course is assigned to a professor because he wanted to teach it, and he decides that he does not care after all whether he teaches it. Whether or not that assignment should be kept in the schedule should be reasoned on the basis of global factors. But the TMS makes the decision syntactically: it would automatically deassign the course if the original preference lost its validity since that preference was used in the reasoning for the assignment.

This problem is exacerbated by including contradiction resolution in the consistency maintenance task of the TMS. This prevents the problem solver from deciding when and how to resolve the contradiction. A more subtle problem arises when context switch rationales are recorded by the elective justification in contradiction resolution.

Suppose that the assignment of Descartes to PHL305 does go OUT. Kant's assignments will be changed automatically by the TMS. They should not be. The effects can ripple up throughout our course schedule revising assignments willy-nilly. This may mean the undoing of a lot of work in developing a good schedule. It may also cause other problems. Suppose that we have published the schedule and promised students that PHL380 will be taught by Kant. Suppose it is later in the year and Kant has prepared to teach PHL380 and not PHL305. There may also later be additional constraints on the amount of change in the plan; e.g, no more than two professors' assignments should be shifted in order to teach the extra section required by increased enrollment. Whether or not to revise Kant's assignment is a *semantic* problem: it must be reasoned. The TMS's automatic, syntactic-based relabeling is not desirable in such cases.

In general, heuristic search guidance in making initial choices represents local optimums. In the case of choice rejections that are no longer valid, the rejected

choice is known to be locally better. But since it is later in the problem solving process, additional reasoning may be necessary to determine whether the locally optimum choice is better globally. Elective justifications supply the information to detect opportunities for reevaluation, but the TMS's automatic labeling denies the opportunity.

An associated problem is that in traditional TMSs, choices are represented as the same kind of assumptions as other sources of nonmonotonicity, including contingencies. Thus a TMS can also be overly-aggressive in resolving a constraint violation, perhaps by assuming that a professor will get a grant when there is no good reason for doing so. This is the "wishful thinking" problem first noted by Morris [13].

Yet if we have made a subplan to purchase textbooks for Descartes' PHL305 class, based on his choice, then the validity of that purchase should go OUT if his assignment to that class goes OUT. And that assignment should go OUT if his unavailability to teach comes IN. Clearly there is valid syntactic work for a TMS to perform. What is it?

An extreme case of a problem solver/TMS architecture that avoids over-aggressiveness is that of Dhar and Croker[5]. Contradiction resolution is entirely removed from the TMS. And the only assumptions possible are choices and there is no justification of the selected or rejected choices other than that they have been directly selected or rejected by the problem solver. Context rationales could be kept but are not. In fact all problem solver/TMS architectures since the ATMS have lacked at least a context switch rationale.[5] Such systems cannot provide the required computational support for the planning problem described here.

How to Make It Work

The definitions of contexts and rationales must be extended beyond the CSP description given earlier to describe a more flexible problem solver. Not all of the details of this system can be provided here. Those not relevant to context maintenance have been omitted.

What's in a Context

It is at least inconvenient, if not practically impossible, to define many problems solely in terms of a state space search of choices of variable value assignments, as with a CSP, or integer programming [4]. In our course scheduling example, sections of courses are generated as needed. Enumerating all possibilities ahead of time is prohibitive, at least in terms of problem formulation, if not computationally.

Thus, let us represent problem solving by goals and operators with the following simple properties: 1) exactly one operator can be applied to a goal at a time and 2) an operator application can result in new subgoals, new variable assignments, or both. This first property implements a single-context search strategy.[6] The second provides a minimal ontology of inferential objects for our purposes, as opposed to *ad hoc* database assertions made by rule firings.

Let us call an operator application a *decision*. We define a set of *admissibility* conditions for each operator such that the decision is *valid* only as long as the conditions are met and the decision is not *retracted*.[7](See Section) This validity can be represented by a TMS justification schema for each decision, shown in Figure 5. The problem solver derives the particular justification for admissibility from those conditions, such as availability. Constraints, as before, are over the variable assignments.

In the course scheduling example, each assignment of a class to a teacher is the result of applying an operator that had at least the admissibility condition that the teacher be available. To switch examples temporarily, in a travel planning problem, an admissible condition for deciding to fly to a destination is that the airport be open. This decision will lead to a subgoal of choosing a flight (even though a particular flight may have suggested the decision) for which any operator will have an admissible condition of the flight not being canceled. Thus are *contingencies* represented.

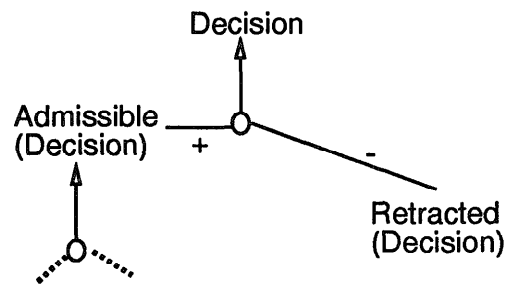

Figure 5: Decision Validity Justification

A CSP is a degenerate case of such a representation in which each goal is to choose a value for a variable, each operator application results in just such an assignment, and there are no admissibility conditions. In a CSP, the validity of each decision corresponds exactly to the validity of an assignment. In this more general architecture, which we call REDUX[19], the validity of multiple assignments and subgoals may be dependent upon the validity of a single decision.

The validity of the decision itself is distinguished from that of its (local) *optimality*. Optimality of a decision represents the heuristic choice of an operator from a conflict set of operators that could have been applied to a goal. This choice depends upon the validity of the goal (which depends in turn on the validity of the decision

[5]E.g., the ATMS provides no rationale for rejected contexts.

[6]Extensions to multiple contexts are possible but have not been studied.

[7]A goals is satisfied if some operator is validly applied to it and all of the resulting subgoals, if any, are satisfied.

that spawned it), the heuristic rationale for choosing the operator (which in turn depends upon the closed world assumption about the conflict set of operators for the goal), and the admissibility of the decision. It also depends upon the decision not being *rejected.* (See Section) Figure 6 shows the TMS justification schema for decision optimality installed by the problem solver for each decision.

Detail of how the problem solver chooses the best operator and creates a justification for BEST-OP is omitted here. Three points are relevant to context maintenance. First, any source of nonmonotonicity in the heuristic choice is reflected in the justification just as *Unless* clauses are in a standard rule-based system. Second, operator instances that have a valid rejection are not chosen. Third, if a normally less preferred choice is made because a better one is rejected, that rejection is included in the justification of BEST-OP.

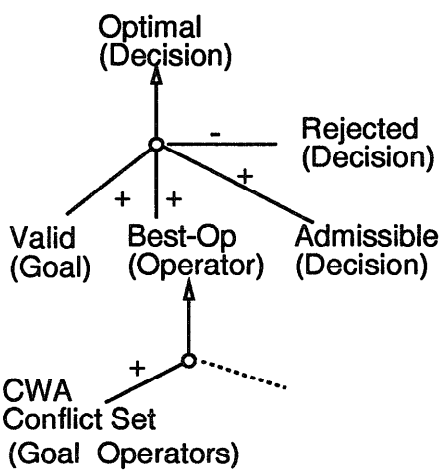

Figure 6: Decision Optimality Justification

We now extend the definition of context from Section : a *context* is the current set of decisions, valid and invalid. The *context rationale* is now the set of decision optimalities, valid and invalid.

Decision Rejection and Retraction

The REDUX problem solver is agenda-controlled. Agenda tasks include satisfying goals, resolving constraint violations, and retracting currently valid but no longer optimal decisions. The last are indicated when the TMS labels IN nodes with justifications that reflect this condition for individual decisions. Which task to perform and how is either reasoned automatically or guided interactively by the user. Of interest here, the problem solver may choose to *reject* a decision to resolve a constraint violation, or because the decision has lost valid reason for optimality, possibly due to a contingency. To do so, the problem solver either adds a premise justification to a retraction, or removes it. (This is equivalent to direct manipulation of the node status by the problem solver.)

There are three simple but fundamental changes that must be made to the standard problem solver/TMS architecture to give the problem solver control over constraint violation resolution.

1) In the standard Doyle-style system, contradiction nodes are distinguished: the labeling algorithm detects the validity of a contradiction and immediately calls a resolution algorithm. If the problem solver is to have control over contradiction resolution in general, contradictions should be put on the problem solver agenda instead. The labeling algorithm should take no other action. In REDUX, contradictions are not distinguished at all; the problem solver tests for constraint violations when updating the agenda.

We now describe two modifications of the Doyle-style implementation of dependency-directed backtracking. The resulting algorithm, together with the data structures described in Section , provides the context rationale for context switches.

2) Given a constraint violation, consider *only* decisions as culprits and their associated retractions as electives. REDUX uses the `FIX` mechanism of [15] to construct the conflict set of underlying decisions as assumptions that might be retracted to resolve the constraint violation. The following `FIX` is used in REDUX is used at the system level, hidden from the user of REDUX primitives:

```
(fix (constraint-violation ?name )
     (decision ?op ?values ?goal)
     (retracted-decision ?op ?values ?goal))
```

REDUX implements the heuristic choice by the problem solver of decisions to retract with the PREFER predicate of [15], with some system-level preferences based on goal hierarchies. For a chosen retraction, a safe, complete elective justification is constructed as described in [15].

3) The second and most important modification of backtracking is not to add elective justification to the retraction, as would normally be done. Instead, for each decision retraction and elective justification, use the justification to justify the decision *rejection* and add a premise justification to the retraction as illustrated in Figure 7. Justifying the retraction has the effect of invalidating the decision and all of the subgoals and assignments that depend upon it. Justifying the rejection has the effect of decision optimality loss. (See Figure 6.) The rejection justification is also used in further problem solving.

The distinction between optimality/rejection and validity/retraction solves the problem of too aggressive TMS relabeling while providing detection of possible opportunities to improve the solution. We can now redraw the justification of Figure 4 using the justification schemata above as shown in Figure 8. Should the decision to assign Descartes PHL305 be retracted, this will leave the decision to assign Kant PHL380 valid but no

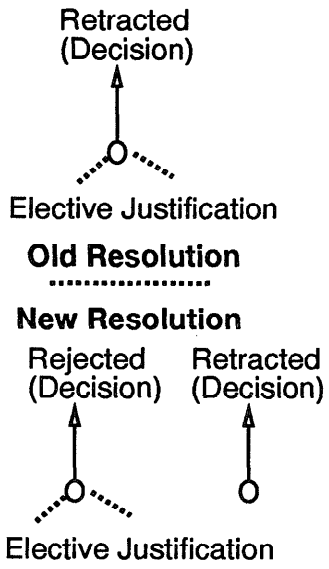

Figure 7: Elective Justification of Rejection

longer optimal. The problem solver now has the opportunity to reason whether the plan should be changed.

Evaluation and Significance

Previous problem solver/TMS models have either not detected possible opportunities for improving the solution in incremental planning or the TMS has been overly aggressive in responding to changes in a plan. The concept of maintaining a context rationale resolves this dilemma.

Truth maintenance formalizations after Doyle have not considered the task of context maintenance [22, 11]. When contradiction resolution has been considered, it has continued to be a function of truth maintenance rather than a search function of the problem solver [21]. We identify context maintenance as a task and show how to modify previous algorithms to implement it.

Complete details of the model are beyond the scope of this paper. In particular, construction of optimality justifications is complex. The rationale for context switches is constructed using the elective justification of [15], which is in turn a revision of Doyle's original "conditional proof" justification [6]. Improvements to these algorithms are described in a forthcoming paper, which will also show that if the elective justification would have been safe to justify the retraction, it will remain a valid justification for the rejection when the retraction is justified by a premise.

Context maintenance is especially important for incremental replanning in response to contingencies. No other technique has been demonstrated that precisely locates the effects of changes in a general plan [12, 27].

The model outlined here has some additional advantages. Truth maintenance is often represented as a

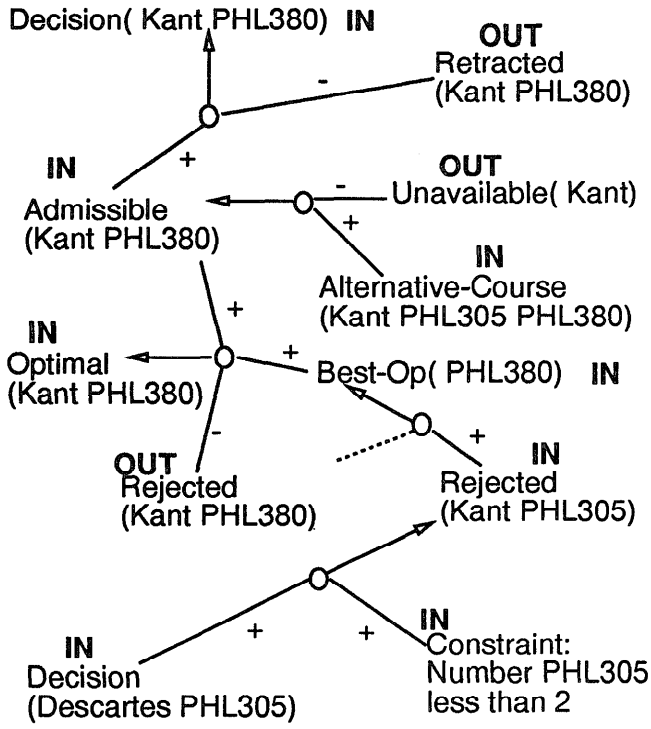

Figure 8: New Alternative Assignment Justification

technique for CSPs: a state-space search problem [3]. The definition of *decisions* in the REDUX architecture allows truth maintenance to be used for problem reduction search as well. An associated idea is justification schemata that define semantics for the dependency network and relabeling behavior. Previous general TMS/problem solver architectures generated *ad hoc* justifications from rules or "consumers" that resulted in complex dependency networks with unpredictable behavior.

The ideas here have been developed during several years of experiments with Proteus applications[25, 4] and implemented in a new system, REDUX [19]. We are currently reimplementing in REDUX two large applications both originally done with different systems [20]. Several extensions, including iterative goals and object creation, have already been found to be necessary. The experiment is to determine whether there is a usefully large class of applications for which the REDUX architecture is feasible and whether it significantly facilitates development.

Acknowledgments: This paper benefits from valuable suggestions by Vasant Dhar, Michael Huhns, and Adam Farquhar. This research is funded by Bellcore.

References

[1] de Kleer, J., "An Assumption-based TMS", *Artificial Intelligence* **28**, pp. 127-162, 1986.

[2] de Kleer, J., "Back to Backtracking: Controlling the ATMS," *Proc. of the Fifth National Conference on Artificial Intelligence*, AAAI, pp. 910-917, 1986.

[3] de Kleer, J., "A Comparison of ATMS and CSP Techniques," *Proc. of Eleventh IJCAI*, August, 1989, pp. 290-296.

[4] Dhar V. and Raganathan N., "An Experiment in Integer Programming," *Communications of the ACM*, March 1990. Also MCC TR ACT-AI-022-89 Revised.

[5] Dhar V. and Croker A., "A Knowledge Representation for Constraint Satisfaction Problems," Dept. of Info. Systems, NYU, Technical Report 90-9, January 1990.

[6] Doyle J., "A Truth Maintenance System," *Artificial Intelligence*, **12**, No. 3, pp. 231-272, 1979.

[7] Dressler O. and A. Farquhar, "Problem Solver Control over the ATMS," submitted to AAAI-89.

[8] Fox M., Sadeh N., and Baykan C. "Constrained Heuristic Search", *Proc. 11th IJCAI*, pp. 309, 1989.

[9] Kumar, Vipin, "Algorithms for Constraint Satisfaction Problems: A Survey," Microelectronics and Computer Technology Corporation TR ACT-RA-041-90.

[10] McAllester D., "An Outlook on Truth Maintenance," A.I. Memo 551, Massachusetts Institute of Technology, AI Lab., 1980.

[11] McDermott, D., "A General Framework for Reason Maintenance," Yale technical report CSD/RR#691, March, 1989.

[12] Morgenstern, L., "Replanning", *Proc. DARPA Knowledge-Based Planning Workshop*, pp. 5-1, Austin, 1987.

[13] Morris, P. H., Nado, R. A., "Representing Actions with an Assumption-Based Truth Maintenance System," *Proc. Fifth National Conference on Artificial Intelligence*, AAAI, pp. 13-17, 1986.

[14] Nilsson, N., *Principles of Artificial Intelligence*, Tioga Pub., Palo Alto, CA, 1980.

[15] Petrie, C., "Revised Dependency-Directed Backtracking for Default Reasoning," *Proc. AAAI-87*, pp. 167-172, 1987.

[16] Petrie, C., et al., "Proteus 2: System Description," Technical Report, MCC TR AI-136-87, 1987.

[17] Petrie, C., et al., "A Planning Problem: Revisable Academic Course Scheduling," Technical Report, MCC TR AI-020-89, 1989.

[18] Petrie, C., "Reason Maintenance in Expert Systems," Kuenstliche Intelligence, June, 1989. Also, MCC TR ACA-AI-021-89.

[19] Petrie, C., "REDUX: An Overview," MCC TR ACT-RA-314-90, October, 1990.

[20] Petrie, C., "Scheduling with REDUX: A Technology for Replanning," MCC TR ACT-RA-340-90, November, 1990.

[21] Reinfrank, M., O. Dressler, and G. Brewka, "On the Relationship Between Truth Maintenance and Autoepistemic Logic," *Proc. of Eleventh IJCAI*, August, 1989, pp. 1206-1212.

[22] Reiter, R., and de Kleer, J., "Foundations of Assumption-Based Truth Maintenance Systems," *Proc. AAAI-87*, pp. 183-188, 1987.

[23] Rossi, F., Petrie, C. and Dhar, V., "On the Equivalence of Constraint Satisfaction Problems", *Proc. ECAI-90*, Stockholm, August, 1990. Also MCC TR AI-022-89.

[24] Simon, H.A., The science of design and the architecture of complexity, in: *Sciences of the Artificial*, MIT Press, Cambridge, 1969.

[25] Steele, R., "An Expert System Application in Semicustom VLSI Design," *Proc. 24th IEEE/ACM Design Automation Conference*, Miami, 1987.

[26] Stefik, M., "Planning with Constraints (MOLGEN: Part 1)," *Artificial Intelligence*, **16**, pp. 111-139, 1981.

[27] Wilkens, D., *Practical Planning*, Morgan Kaufmann, San Mateo, 1988.