An Approach to Dependency Directed Backtracking using Domain Specific Knowledge

Vasant Dhar
Dept. of Computer Applications and Information Systems, New York University.
Casey Quayle
Bolt, Beranek and Newman Inc., Boston, Massachusetts.

Abstract

We describe a problem solver that met non-chronological backtracking in situations involving tradeoffs. The novel aspect of the problem solver is it ability to weigh advantages and disadvantages of alternatives at choke points. Whenever untenable situations arise, this information is available to the backtracker to determine the most appropriate backtracking point. By endowing the backtracker with access to domain-specific knowledge, a highly contextual approach to reasoning in dependency directed backtracking situations can be achieved.

An area of investigation now commonly referred to as "Dependency Directed Reasoning," has had a major impact on AI research in the last decade (de Kleer et.al, 1977; Doyle, 1978, 1979, 1980; McAllcster, 1980; Stallman and Sussman, 1977). In this paper we describe some dependency directed reasoning features of a problem solver called PLANET (Dhar, 1984). It has been designed to help planning managers in a large computer manufacturing company with the formulation and investigation of models for allocation of resources. Since the process of resource planning[1] involves making assumptions that are continually subject to revision, dependency information plays a crucial role in the maintenance and incremental change of planning models. What is of particular interest in this paper is a heuristic procedure for dependency directed backtracking that addresses one drawback of existing dependency frameworks, namely, determining what belief (set of assumptions) in an existing model to change whenever an undesirable state arises.

2. The Need far Dependency Directed Backtracking

The problem of formulating a resource planning model has two important features. First, as assumptions about various parts of the task environment are made, choices in other parts of the environment are constrained. Second, there are usually resource requirement tradeoffs among the alternatives that can be made. These features are operationalized in PLANET in terms of a "compute-quiesce-guess" cycle as in MOLGEN (Stefik, 1980) where processing continues with existing constraints until a quiescent state is reached. When the program quiesces, but has not completely solved the problem, it creates a new constraint by "guessing" and restarts processing. The guess is based on a heuristic evaluation of alternative choices. The cycle of compute-quiesce-guess continues until the problem is solved.

Table 1 shows various resource requirements for carrying out four manufacturing activities. For example, resource requirements for "S-test," which an activity that tests assembled modules, depends on the choice of testing devices. In choosing among such alternatives, the program picks the most "balanced** alternative in light of the organization's resource availability picture at the time. Because these choices are made successively using limited look-ahead,[2] it leaves open the possibility that some resource constraint will be violated, thereby forcing the problem solver to undo one or more of its previous choices.

As an example, let us consider a situation where the three choices indicated in table 1 have been made when saving space was considered more important than saving capital. Further, assume that at this point in the plan formulation, there is only $2 million more left to be allocated. With this "money is no object** attitude the program is now in trouble. It cannot chose either the SIM-tester or the FA-tester, as either choice would consume more than the remaining capital. Clearly, some previous selection must be undone to alleviate the problem and several maneuvers are available. Under such circumstances, the problem solver must be capable of reasoning about the most rational course of action rather than simply making a blind selection. In the following paragraphs, we present a formal treatment of PLANETs backtracking method.

3. Dependancy Directed Resoning in PLANET

The data dependency and backtracking mechanisms of PLANET are built on top of RUP (McAllester, 1980). An overconstrained state is equivalent to a RUP contradiction. When overconstrained, PLANET must find and retract an existing choice that contributed to die unacceptable state of affairs. An alternative choice is then made as a

[1] We use the term paw and *planning* to refer to *busbmt* (manufacturing or resource) plan(ing), and not a plan as normally understood in AI. A manufacturing plan it an interrelated set of choices about various parts of the manufacturing process. A resource plan indicates the types and amounts of resources a manufacturing plan requires.

[2] It does not know how many choices still need to be made, i.e. how much of the model stilt needs to be crafted, and what me tradeoffs involved will be.

Table 1

These tablet indicate resource-requirements/tradeoffs for four decisions. The selected alternative in the first three is in bold type. The units of Labor are workers working, Capital is in millions of dollars, and Space is in thousands of square feet of floor space.

	Labor	Capital	Space
Activity: MIF-Test			
Shorts/opens-tester	**1**	**3**	**2**
DL	**3**	**0.5**	**3**
Activity: S-Test			
QV	**1**	**2**	**4**
L-20	**2**	**6**	**2**
FC-33	**1**	**3**	**3**
Activity: Insertion			
P1	**2**	**4**	**1**
P2	**2**	**2**	**2**
Manual	**5**	**0.5**	**3**
Activity: Peripherals-Test			
SIM-Test	**3**	**4**	**3**
PA-Test	**3**	**3**	**6**

replacement, and processing continues. We refer to this backtracking behavior as Second Guessing.

Whenever the program is quiescent and must resort to guessing it is making a *"Forced Choice".* In such situations, several alternatives typically exist. We call each forced choice a decision level item. Each of the alternatives available in the context of a decision level item is called a selection level item. Note that for any given decision level item there exists a set of at least two selection level items. For exposition, we will denote particular decision level items with a subscripted D and denote particular selection level items with a subscripted S. Sets of either will be denoted in italics, i.e. D, and 5.

When the program is processing a decision level item it is guessing. The guess is to pick some element from a set of selection level items. When the guess is made, only heuristic estimates of resource tradeoffs are available to evaluate the merits of particular alternatives. By the time the program finds itself in an over-constrained state considerable computation could have been done in developing a more accurate assessment of resource consumption and establishing other constraints. This information is not available to PLANET when guessing but is available when second guessing. The key point here is that the evaluation function used by the second guesser is more accurate than the evaluation function used by the guesser, allowing problem solving to continue with the benefit of new hindsight.

When the program becomes over-constrained it has processed several decision level items. We call the set of these items $\mathcal{D}$. For each $D_t \in \mathcal{D}$ there is a set, S_t of alternative selection level items (choice) that could have been guessed. We call the distinguished guess in S, as S_{ct}. Thus, the set of all selection level choices (guesses) that have been made is $\$ = \{S_{ct}\}$, for all t.

A subset, S , of \$ contains all S guesses that have contributed to the over-constrained state. 5 can be computed by chasing current dependency information. The subset, D, of b - the set of decision level items that need to be reconsidered - is derivable from 5.

Given D, the second guesser can reason, in a "given what I know now" manner, about two related issues:

Determize which element of D to reconsider.

Reconsider, find a different selection for the chosen decision item.

To elaborate, consider the previously described situation where PLANET has suddenly realized that it is about to over allocate capital by at least \$1 million. Prom table 1, we see that the set 5 is {shorts/opens-tester, L-20, PI). The set D is then {MIF-test, S-Test, Insertion). Examination of D reveals that S-Test is the best decision level item to reconsider. The set {L-20 QV FC-33) is then scrutinized using the added information that an acceptable choice would have to consume at least \$1 million less than L-20. Note that in second guessing, PLANET has access to information that was not available when it first attempted to process the decision level item S-Test. Quite literally, it is using hindsight.

The set of choices in 5 are indeed responsible for the over constrained state. We claim that the identification of a scapegoat in S and its replacement is made more rational by examining each selection level candidate in the context of its decision level item. Furthermore, each of the decision level items should be compared and contrasted. This distinction is important - the selection level items art the retractable "assumptions", but second guessing pivots around the decision level items.

4. Reasoned Assartien and Retraction— of Assumptions

In order to facilitate a rational reconsideration of choices, we provide explicit structures that maintain context at the decision level. Whenever a contradiction arises, the program can use this plus information on the type of constraint violation to determine which assumptions to retract to best alleviate the problem. Each choice is represented as a structured object (enclosed within braces):

{decision: D_t
selection: S_t
alternatives : $\{S_1, \ldots, S_n\}$
disadvantages : $((S_1(R1R2..Rn))(S_2(R1R2\cdots Rn)))$
advantages : $((S_1(R1R2..Rn))(S_2(R1R2\cdots Rn)))$]

The "advantages" and "disadvantages" slots are lists of dotted pairs; the first element in a pair is a selection item (an alternative to the selection S_1), and the second is a list of resource categories for which the alternative is

advantageous or disadvantageous respectively. Whenever an unacceptable allocation of resource, R, arises, the program invokes a two step procedure to determine its revised set of choices. First, all decision level items disregarding R as a disadvantage are recorded. Then, the combined pool of selection level items are compared to determine the selection best alleviating the underlying problem. The object representation of the S-Test decision in our example would be:

{decision: (modules: S-Test)
selection: L-20
alternatives : (QV FC-33)
disadvantages : ((QV labor capital)(FC-33 labor capital))
advantages : ((QV space)(FC-33 space))}

In this example, PLANET correctly identified three decision points as having disregarded capital. It identifies S-Test as the decision level item to reconsider, and modifies the selection associated with it. The important aspect of this examination is that it had a specific goal: Identify the decision level items that contribute to the over constrained situation that also disregard the down side on the resource for which a constraint has been violated. In effect, the procedure focuses on a problem specific set of contributing factors instead of merely finding some basic set of entailment.

PLANET routinely makes these kinds of revisions. When faced with the task of reevaluating previous choices, it applies both problem specific and domain specific knowledge to identify the choice. In doing so, it uses hindsight - the current problem state - to identify its backtracking point. Like other non-chronological backtrackers (Stallman and Sussman, 1977; Doyle, 1979; McAllester, 1980) only those inferences dependent on the choice are retracted.

In the example considered above, a detailed comparison of the resource implications of alternative selections was possible. Unfortunately, this is not always the case. If the set of potential backtracking points identified includes choices about the more abstract parts of a manufacturing plan where detailed resource requirements have not yet been assessed, a quantitative comparison is not possible. Projecting the consequences of the various maneuvers is then difficult. Further, if long chains of dependencies exist, the program might pick as its best move one that involves undoing large parts of the partially formulated plan since this would free up the maximum amount of the scarce resource. In other situations, a revision might only alleviate the problem marginally, allowing it to recur a few steps later. Finally, a limitation of the existing backtracking scheme is that the program is unable to recognize situations where it might be better off retracting a *combination* of choices as opposed to a single decision. We are currently working on ways by which domain specific knowledge about these situations may be represented in terms of dependency information and made accessible to the backtracker.

In conclusion, the issues raised here have been driven by a complex, real-world problem where existing formalisms proved to be useful but inadequate in modeling the essential nature of the problem. We have developed a scheme whereby a backtracker might assess more rationally the reasons for an untenable situation, and modify its existing set of choices in light of the evolving scenario of constraints. While the methods outlined above arc preliminary, they represent a step toward a more general method for reasoned introduction and retraction of assumptions for decision situations where tradeoffs are involved.

References

1. de Kleer, J., Doyle, J., Steele, G. and Sussman, G., AMORD : Explicit Control of Reasoning, Proceedings of the Symposium on Artificial Intelligence and Programming Languages, 1977.
2. Dhar, Vasant., PLANET: An Intelligent Decision Support System for the Formulation and Investigation of Formal Planning Models, Ph.D. Thesis, University of Pittsburgh, 1984.
3. Doyle, Jon., Truth Maintenance Systems for Problem Solving, TR-419, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, 1978.
4. Doyle, Jon., A Truth Maintenance System, *Artificial Intelligence,* June, 1979.
5. Doyle, Jon., A Model For Deliberation, Action, and Introspection, MTT-AI TR-S81, May 1980.
6. McAllester, D., Reasoning Utility Package, AI Laboratory Memo 667, April 1982.
7. McDermott, Drew, and Doyle, Jon., Non-Monotonic Logic I, *Artificial Intelligence* Not 1 and 2, April 1980 (special issue).
8. Stallman, Richard, and Sussman, Gerald., Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis, *Artificial Intelligence,* volume 9, No.2, October 1977, pp 135-196.
9. Stefik, Mark., Planning with Constraints, Stanford University, Computer Science Department, STAN-CS-80-784,1980