

Passive and Active Decision Postponement in Plan Generation

David Joslin*

Martha E. Pollack^{†,*}

*Intelligent Systems Program

[†]Department of Computer Science

University of Pittsburgh, Pittsburgh, PA 15260

joslin@cs.pitt.edu, pollack@cs.pitt.edu

Abstract. One of the strengths of Partial-Order Causal Link (POCL) planning is the ability to postpone decisions. But because postponed decisions play no role in reasoning about the plan until they are eventually acted upon, the penalty for postponing some decisions can be quite high. We call this style of decision postponement *passive postponement*, and present experimental results that quantify the efficiency penalty it incurs. We also suggest an alternate approach, *active postponement*, that allows postponed decisions to impose constraints on the generation of a plan. This constraint-based approach to decision postponement has been implemented in the Descartes planning system. In Descartes, every planning decision is represented by a variable, with constraints on each variable representing criteria that must be satisfied by the corresponding decision. These constraints are managed by a general-purpose constraint engine, so that even postponed decisions play a role in reasoning about the plan.¹

1 Introduction

Modern Partial-Order Causal Link (POCL) planners such as SNLP [10] or UCPOP [12] often have difficulty with problems that really ought to be trivial. For example, in experiments we conducted using a highly simplified TileWorld domain (a dynamic gridworld) [16, 17], even the generation of a simple plan to pick up two tiles and fill two holes turned out to be intractable for UCPOP. Various formulations of the domain, along with a variety of domain-independent search heuristics, were all unsuccessful. In this case, it turned out to be easier to write a special-purpose planner for TileWorld.

But why should such an apparently simple problem cause so much difficulty in the first place? To investigate the sources of this difficulty we conducted a series of experiments, some of which were reported in [6]. In this paper we present new experimental results that further clarify a key source of difficulty for traditional planning techniques, namely, their reliance on ineffective methods for decision postponement. Essentially, the problem is that POCL planners postpone decisions (about how to achieve certain

¹This work has been supported by the Air Force Office of Scientific Research (Contract F49620-91-C-0005), by the Rome Laboratory (RL) of the Air Force Material Command and the Advanced Research Projects Agency (Contract F30602-93-C-0038), by an NSF Young Investigator's Award (IRI-9258392) to Prof. Martha Pollack, and by a Mellon pre-doctoral fellowship to David Joslin.

goals, the order in which to perform planned actions, and so on), and, once postponed, these decisions play *no* role in the plan generation process until they are selected for consideration. As a result, the penalty for postponing some decisions can be quite high. We call this approach *passive postponement*.

In this paper, we show how these difficulties can be avoided by a new approach *active postponement*, in which even postponed decisions play a role by constraining the plan being generated. This technique has been implemented in the Descartes system.

In Descartes, planning problems are transformed into Constraint Satisfaction Problems (CSPs), and then solved by applying both planning and CSP techniques.² In addition to addressing efficiency issues, the Descartes system also extends representational flexibility of traditional POCL algorithms in two ways. First, it represents plans using a Temporal World Model (TWM) similar to that of Allen and Koomen [1]. This allows Descartes to handle goals with deadlines and arbitrary temporal relations between preconditions and effects. Second, the constraint representation makes it possible to handle not just conjunctive goals, but also goals involving disjunctions or other logical relations. In addition, preconditions and effects may be conditionalized on arbitrary boolean expressions. These and other representational extensions follow very straightforwardly from the application of standard CSP techniques to planning decisions, with the resulting ability to handle arbitrary constraints.

Because of space limitations, this paper focuses only on a comparison of active and passive postponement, and only briefly summarizes some of the other contributions made in the Descartes approach to planning. Further details are available in [5].

The following section describes the passive postponement approach commonly used in POCL planners, explains why that approach can be a source of inefficiency, and presents experimental evidence of the problem. Section 3 describes active postponement, as implemented in the Descartes system. Section 4 gives a brief summary of some of the other advantages of the Descartes approach. The final two sections review related work, and summarize some directions for further research.

2 Passive postponement

There are undoubtedly many factors that can make a simple problem intractable for a given planning algorithm. As several studies have shown [6, 15, 11], POCL planners are very sensitive to the order in which decisions are made. In particular, premature commitment to planning decisions can result in state space explosions. The idea behind “least commitment” planning is to postpone decisions until they can only be made in one way, i.e., they are forced; the ability to postpone unforced decisions is one of the strengths of POCL planning algorithms.

But the postponement of decisions can also be a source of inefficiency. We draw a distinction between two techniques for postponing decisions—active and passive postponement—and show how the wrong approach to postponement can result in large efficiency penalties.

As with many planning algorithms, POCL planners perform a best-first search through the space of partial plans. Each partial plan has a set of *flaws* to be repaired, where a flaw is either a goal to be achieved or a threat to be removed. The

²In general, a planning problem cannot be transformed into a single static CSP. Descartes can be thought of as transforming a planning problem into a *dynamic* CSP to which new constraints and variables are added during the solution process. The dynamic CSP is solved by breaking it down into static CSPs, to which standard CSP techniques may be applied. For further details see [5].

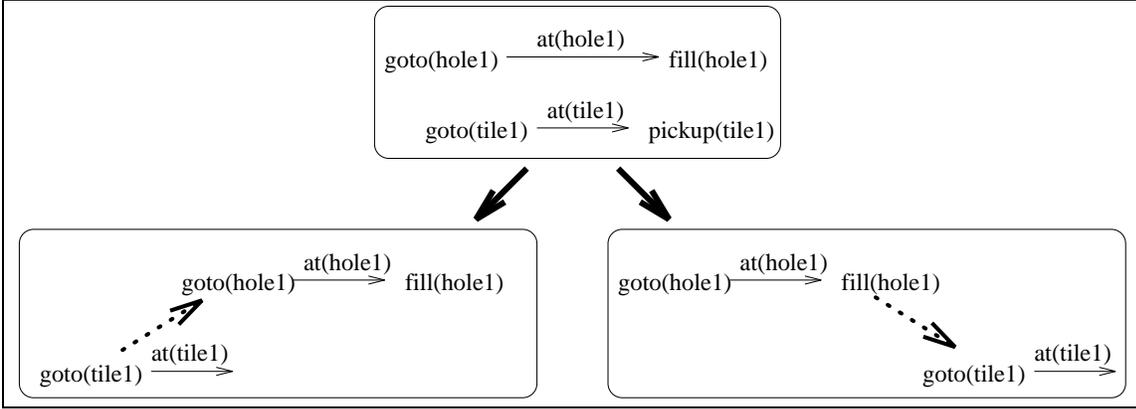


Figure 1: Threat resolution in POCL planners

flaws in the root node represent the goals for which a plan is being generated.

POCL algorithms proceed by *refinement search* [7]. A node (a partial plan) is selected, then a flaw is selected within that node; successor nodes are then generated for each possible repair of that flaw. New flaws may be introduced as new steps are added, with preconditions that must be achieved and/or new threats that must be removed. The criterion of success is that a node have no flaws.

To illustrate this algorithm in the case of threats, consider the example in Figure 1. Three (simplified) nodes in the search space of a TileWorld problem are shown here. The top node gives an example of a threat, and the two successor nodes show how that threat could be resolved.

In the TileWorld domain, the agent moves around a grid, picking up tiles and dropping them in holes. In the top node in the example, the $fill(hole1)$ action has a precondition of $at(hole1)$, and the $goto(hole1)$ action is being used to satisfy that precondition. Another action, $goto(tile1)$, has been added to satisfy another precondition, $at(tile1)$, but no decision has been made yet about exactly when that action will occur. The $goto(tile1)$ action threatens to undo the $at(hole1)$ precondition, because the agent cannot be in both places at the same time.³

The two successor nodes in Figure 1 show how a POCL planner would resolve this threat. The dotted arrows represent temporal orderings: the $goto(tile1)$ action must occur either before $goto(hole1)$, or after $fill(hole1)$. When this threat is selected for repair, successor nodes are generated for each of these two options. In some cases there may be additional options for resolving a threat, by imposing binding constraints on plan step parameters.

When a flaw is selected and repaired, all other flaws in that node are simply passed on to any successor nodes. Each flaw in a node represents a decision to be made about how to achieve a goal or resolve a threat; each unselected flaw that is passed on to the successor nodes represents a postponed decision. For example, in Figure 1, if the parent node has flaws other than the selected threat, then the successor nodes that repair the selected threat will also inherit those other flaws.⁴

³Intuitively we have a threat because the agent cannot be in both places at once. In actuality, only threats between propositions and their negations, i.e., between $at(hole1)$ and $\neg at(hole1)$, are recognized. The details of how this works out in this particular example are not important to the issues raised here.

⁴Sometimes repairing one threat may also happen to repair another threat as well; the other threat will still be inherited, but will be discarded as soon as it is selected and recognized as having been

We term this approach to postponing decisions *passive postponement*. Decisions that are postponed in this manner play no role in reasoning about the plan until they are actually selected. Passive postponement of planning decisions can incur tremendous performance penalties. It is easiest to see this in the case of a node that has an unrepairable flaw; such a node is a dead end. But the node may not be *recognized* as a dead end if some other (repairable) flaw is selected instead, i.e., the decision about the unrepairable flaw will be postponed. In that case, one or more successor nodes will be generated, each inheriting the unrepairable flaw, and each, therefore, also a dead end.⁵

In this manner, a single node with a fatal flaw may generate a large number of successor nodes, all of them dead ends. That dead-end region of the search space could have been pruned in one step, had the fatal flaw been selected when it was first generated; instead, each of the successor nodes must be visited before the dead-end region is pruned.

The propagation of dead-end nodes is an instance of a more general problem. We can think of postponed decisions as placing constraints on the plan generation process; with passive postponement those constraints are not taken into account. A fatal flaw is simply a constraint that cannot be satisfied. Failing to take other flaws into account can also be a problem. Suppose that a node has a flaw, f , that can only be repaired in one way. If that flaw is selected, there will be only one successor node. If, however, other flaws are selected first, the forced repair of f will have to occur in each successor node, rather than just once. The time spent resolving f is thus multiplied. Passive postponement also means that interactions between the constraints imposed by N postponed decisions are not recognized until all N decisions have been selected.

2.1 *Experimental results*

The propagation of dead-end nodes is not just a theoretical problem; it can be shown experimentally to cause serious efficiency problems for POCL planners. The TileWorld domain, mentioned above, was an early motivation for this investigation, and provides a good example. Many TileWorld problems that seem as if they should be very easy turn out to take an extremely long time for a POCL-planner like UCPOP. For example, one TileWorld problem we studied⁶ was not solved by UCPOP, even using a variety of domain-independent search control heuristics and allowing the program to run for over twenty-four hours. A closer examination showed that, in fact, a small number of nodes with fatal flaws (approximately ten) had been multiplied into thousands of dead-end nodes. Although some nodes were eventually recognized as dead-ends when the fatal flaw(s) they included were selected, dead-end nodes were being generated more quickly than they were being detected.

To see how general this problem might be, we ran UCPOP on the same test set of 49 problems from 15 domains used in [6], using the default search heuristics provided by UCPOP. As in the earlier experiments, thirty-two of these problems were solved by UCPOP, and seventeen were not, within a search limit of 8000 nodes generated. Without changing the algorithm itself, we counted the number of nodes examined, and the number of those nodes that were successors of dead-end nodes, i.e., nodes that

resolved.

⁵In POCL planners, an unrepairable flaw cannot become repairable as a result of other decisions.

⁶The goal was to fill two holes. The final plan required only eight steps: go to the first tile, pick it up, go to the first hole, drop the tile in the hole, then do the same for the second tile and hole.

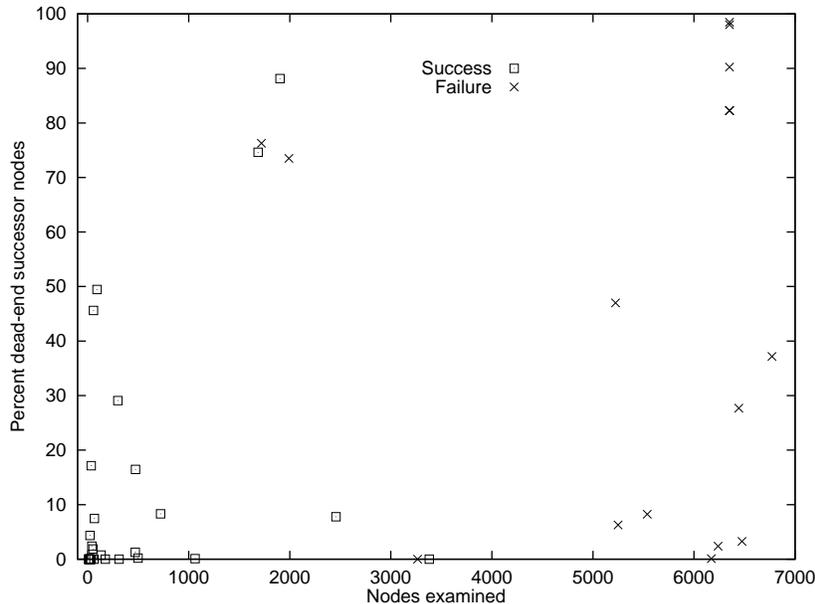


Figure 2: Effort wasted on dead-end nodes

would never have been generated if fatal flaws were always selected immediately.

Figure 2 is a scatter plot, with one point for each of the 49 problems. The x -coordinate of each point is the number of nodes that had been examined when a solution was found or the search limit was reached. (The search limit is on the number of nodes *generated*; the number of nodes examined will be smaller.) The y -coordinate is the percentage of the nodes generated that were successors of dead-end nodes. This latter figure can be seen as a lower bound on the percentage of the search that was wasted effort as a result of passive postponement. It is only a lower bound because, as we mentioned earlier, passive postponement can result in other forms of inefficiency as well. The symbol used to plot each point indicates whether or not the problem was solved: “X” denotes failure, and a box denotes success.

For some problems (see the upper-right portion of the graph), the planner spends as much as 98 percent of the search examining successors to dead-end nodes. Even some successful problems waste eighty to ninety percent of the search in this manner; the average for successful problems was 24 percent, and for unsuccessful problems, 48 percent. We can also see that the propagation of dead-end nodes is clearly not the only source of difficulty, since some failed problems (lower-right portion of the graph) spend little or no time visiting successors to dead-end nodes.

2.2 Smarter passive postponement

One response to this problem is to continue passive postponement, but to be smarter about which decisions are postponed. This is one way to think about the Least-Cost Flaw Repair (LCFR) flaw selection strategy we presented in [6]. For LCFR, we define the repair cost for a flaw to be the number of successor nodes generated, and always select a flaw that minimizes the repair cost. Thus, a fatal flaw has a repair cost of zero (no successor nodes will be created because there is no way to repair the flaw), and will always be selected before any nonfatal flaw. Thus, no successors will ever be created for dead-end nodes. The LCFR strategy also prefers flaws that can only be repaired in

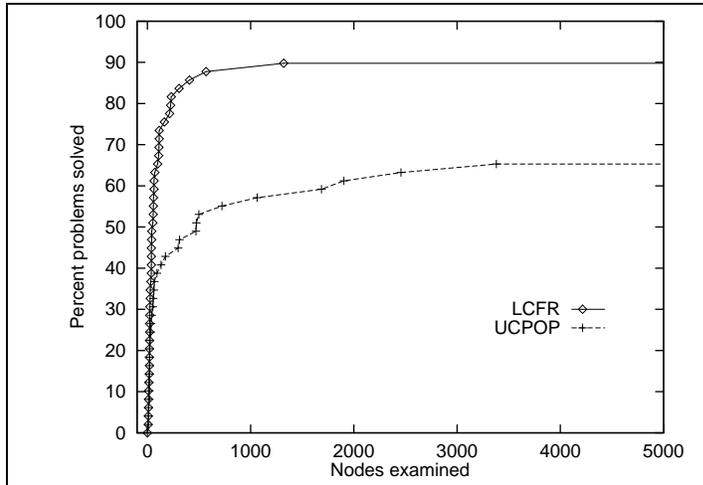


Figure 3: Comparison of UCPOP and LCFR

one way to flaws that will generate multiple successor nodes.

The LCFR strategy is similar to one of the search heuristics used in the O-Plan system [4]. Other related strategies are DMIN and DUnf [14], and ZLIFO [19]. LCFR is also closely related to the “smallest domain first” strategy for selecting variables in constraint satisfaction algorithms [23].

Figure 3 shows the percentage of problems solved by UCPOP and LCFR within a fixed number of nodes examined. (Each point $\langle x, y \rangle$ indicates that $y\%$ of the 49 test problems were solved by examining x nodes or fewer.) As expected, LCFR searches significantly fewer nodes than UCPOP.

Being “smarter” about which planning decisions to postpone is clearly a step in the right direction. However, even with an LCFR-style strategy, decisions are postponed passively, i.e., they do not play a role in reasoning about the plan until they are selected. Moreover, LCFR can only recognize dead-ends in which some flaw is immediately unreparable, i.e., if that flaw is selected, it will have zero successors. If two flaws interact in such a way that either can be repaired, but not both, then that interaction will only be recognized after further expansion of the search tree has occurred. A potentially better approach is to perform what we call *active postponement*, in which postponed decisions are still influential in plan reasoning.

3 Active postponement

Flaws represent decisions that have to be made about how a goal is to be achieved, or how a threat is to be resolved. In making these decisions certain constraints must be satisfied. For example, the decision about resolving the threat in the TileWorld example in Figure 1 must satisfy constraints that ensure that the threatening step does not occur over a certain interval. When the flaw representing this decision is selected, successor nodes are generated that are guaranteed to satisfy those constraints. But until that flaw is selected, no reasoning is done about those constraints.

Active postponement is a constraint-based approach to planning that allows even postponed decisions to impose constraints. When a flaw is first introduced, a constrained variable is added to the plan, representing the possible ways of repairing that flaw; the actual decision about which repair to use can be made at any point by bind-

ing that variable. The introduction of new constrained variables is what makes this a *dynamic* CSP. A general-purpose constraint engine enforces the constraints that must be satisfied in order to repair the flaw.

To illustrate how this works, consider again the example in Figure 1. As soon as the threat is introduced, we know that at some point we are going to have to decide either that $goto(tile1)$ occurs before $goto(hole1)$, or that it occurs after $fill(hole1)$. Our active-postponement planning systems, Descartes, introduces a constrained variable to represent this decision:

$$D_t \in \{a, b\}$$

and posts the following constraints:

$$(D_t = a) \rightarrow after(goto(tile1), fill(hole1))$$

$$(D_t = b) \rightarrow before(goto(tile1), goto(hole1))$$

The two choices correspond to the two successor nodes in Figure 1. The decision about how to resolve the threat can then be postponed, and may be made at any time by binding the variable D_t .

Such constraints can be managed by a general-purpose constraint engine; this is what allows a postponed decision to play a role in reasoning about the plan. For example, suppose that at some point after the above constraints have been posted, it is determined that it is impossible to satisfy $after(goto(tile1), fill(hole1))$, perhaps because $tile1$ is to be used to fill $hole1$, and is consumed in the process. From this, the constraint engine can deduce that $D_t \neq a$, and thus $D_t = b$, and thus $goto(tile1)$ must occur before $goto(hole1)$. This chain of reasoning happens automatically in the constraint engine. Similarly, if the threat becomes impossible to satisfy, the constraint engine can immediately deduce that the constraint $D_t \in \{a, b\}$ cannot be satisfied, and that the current node is therefore a dead end.

Active postponement of goal achievement is more complicated than active postponement of threat resolution, but the idea is the same: post the constraints that must be satisfied in order for a goal to be achieved, then postpone decisions about how to achieve the goal. Suppose, for example, that the goal of $at(hole1)$ might be achieved by either a $goto(hole1)$ operator or a $jumpto(hole1)$ operator. Suppose further that we have $at(hole1)$ as a precondition of a $fill(hole1)$ action. At some point we will have to decide which of the two possible operators will have to be instantiated and used to achieve that precondition. As with threats, Descartes introduces a constrained variable to represent this decision:

$$D_g \in \{g, j\}$$

The two possible operators are *tentatively* instantiated in the plan, and two new boolean variables, G and J , are introduced to represent the decisions of whether or not these $goto$ and $jumpto$ steps are actually in the plan. For example, if G is false then the $goto$ operator is not included in the plan; if G is true, then the $goto$ operator is actually (not just tentatively) in the plan. The variable G refers only to one particular instance of the operator; even if G is false, there may be other instances of the $goto$ operator that are included in the plan.

The following constraints must then be satisfied in order to achieve the goal:

$$(D_g = g) \rightarrow G \wedge before(goto(hole1), fill(hole1)) \wedge \dots$$

$$(D_g = j) \rightarrow J \wedge before(jumpto(hole1), fill(hole1)) \wedge \dots$$

The decision between *goto* and *jump* can then be postponed, and may be made at any time by binding the variable D_g . Binding $D_g = g$, for example, will force G to be true, forcing the *goto* operator to be included in the plan, and will also impose an ordering between the *goto* and *fill* operators. (This example is simplified, and other consequences may also apply.)

As before, the constraint engine may make deductions about these deferred decisions as planning progresses. If, for example, the *jump* operator has a precondition that cannot be satisfied, then the constraint engine may be able to deduce that the *jump* operator cannot be instantiated into the plan, i.e., the boolean variable J is forced to be false. This forces G to be true, with the consequences described above.

The obvious question is to what extent active postponement increases planning efficiency. We would like to be able to compare active postponement with POCL planning and with techniques such as LCFR. Although the Descartes system has been implemented, this paper reports on work in progress, and formal experiments are not complete.

We are, however, able to report results on a few specific problems. The TileWorld problem that was intractable for UCPOP—unsolved even after twenty-four hours on a MIPS Decstation—was solved by Descartes in under three seconds on the same machine. In this particular case, the primary efficiency improvement appears to be from the ability to recognize dead ends immediately.

Most of the other test problems used in the development of Descartes have taken advantage of its representational extensions, and are therefore difficult to compare directly with POCL planners. One example of a problem that can be handled by the current implementation is based on the DIPART [18] domain, and involves trucks that travel between cities. A truck has a gas tank that can be filled at a rate proportional to the time spent at the gas pump. The time required to drive from one city to another is proportional to the distance between them, as is the amount of gas consumed. Three goal conditions must be met: city A must be visited by a certain deadline, city B must be visited by a different deadline, and there must still be a certain amount of gas remaining in the tank when the trip is complete. The solution to such a problem is a schedule of actions, showing when and for how long each action occurs. This problem was solved in 12 seconds. Without other reference points, this performance serves primarily to demonstrate the feasibility of the approach.

One particular efficiency concern in this approach to planning is that each node in the search contains a CSP. To avoid having to maintain copies of each CSP on the frontier of the search, the current implementation maintains one “base node” to which all new plan steps are *tentatively* instantiated. Nodes in the search are then generated, as needed, by applying and propagating constraints that can exclude some of the tentative steps, as well as other constraints that define the node. This helps to avoid excessive memory consumption.

4 Dual-representation planning

Descartes implements active postponement by representing every planning decision with a constrained variable, and posting constraints that represent the correctness criteria for the plan. The examples in the previous section give a general idea of how these variables and constraints are generated.

The variables and constraints, taken by themselves, describe a constraint satisfaction problem. As with any CSP, the problem is solved when all of the variables have

been bound in such a way that all of the constraints are satisfied. Because Descartes represents plans as CSP problems, it can “solve” them, i.e., generate plans, by using standard CSP techniques. At the same time, the plan representation of the problem is also available to Descartes, and planning techniques, including techniques from POCL planning algorithms, can also be applied. The ability to view a problem as either a planning problem or a CSP gives rise to what we call *dual-representation planning*. Due to space limitations, here we can give only a very brief overview of the Descartes approach to planning; for more details, see [5].

Temporal world models. Like many other planning algorithms, Descartes performs a best-first search through the space of partial solutions to a planning problem. Each node in this search space includes both a partial plan and a corresponding CSP. The partial plan consists of a set of plan steps and a set of flaws to be repaired, and differs from the representation used in other POCL planners primarily in that a Temporal World Model (TWM) representation is used.

The TWM used in Descartes follows that of Allen and Koomen [1]. Each proposition, such as *at(tile1)*, has an associated temporal interval over which that proposition is asserted. These are known as Temporally Qualified Assertions (TQAs).

A plan step consists primarily of a set of TQAs, and a set of constraints on those TQAs. Temporal constraints, mathematical constraints, and boolean constraints may be combined. This allows Descartes to provide a great deal of representational flexibility. Temporal constraints, for example, allow problems to combine elements of planning and scheduling, along with other advantages, some of which were summarized in the introduction.

The CSP view. Each node in the Descartes search space includes both a plan representation, using a TWM, and a corresponding CSP. Both views share a common set of variables representing planning decisions. Whenever a plan step is added to a node, new variables and constraints are added as described for active postponement. The shared variables plus the constraints describe a constraint satisfaction problem.

This CSP can be solved by applying standard CSP techniques, but the plan view can also be used, so that standard planning techniques may be applied as well. The Descartes algorithm guarantees that if the CSP view is solved, then the shared variables will be bound in such a way that the resulting plan in the plan view will be a solution to the planning problem. Similarly, any solution to the planning problem will bind shared variables in such a way that the corresponding CSP is also solved.

Decision strategies. The two views of a problem may be used in very flexible ways by a “decision strategy” in the process of finding a solution. The core of the Descartes algorithm itself does not specify how or in what order decisions are to be made. Rather, by representing all decisions uniformly as variables, and by posting constraints that enforce the plan correctness criteria, *any* decision strategy implemented in this framework will create a sound planning algorithm.⁷

Within the Descartes framework a wide variety of strategies can easily be implemented and compared. Decision strategies can flexibly combine search, partial commitment, and CSP techniques. To illustrate the flexibility available, it would be possible to implement a very close approximation of SNLP; such a decision strategy would make all

⁷Any decision strategy that follows certain extra rules will be guaranteed to create a complete planning algorithm.

of its decisions based on the plan view. It would also be possible for a decision strategy to focus on the CSP view, consulting the plan view only for the addition of plan steps.

Decision strategies that combine CSP and planning techniques are also possible, and show the most promise. Traditional planning approaches benefit from being able to make use of constraint satisfaction techniques. At the same time, CSP approaches can be combined with a best-first search through the space of partial solutions, and can make use of semantic information available only from the planning view of the problem.

The flexibility available for decision strategies makes the Descartes framework very useful for experimental comparisons of different approaches. Descartes is particularly useful for investigating least-commitment strategies, because early commitments are not forced for any decisions of any type.

Theoretical analysis. The Descartes approach also makes possible a better intuitive and theoretical understanding of planning problems and algorithms. By representing every decision as a variable to be bound, details that are hidden in traditional planning algorithms become clear. For example, in [5] the CSP formulation is used to give a precise, mathematical characterization of “early commitment” and “least commitment” decision making, including a quantifiable definition of varying degrees of commitment. Least-commitment decision making has typically been taken to mean that decisions are postponed until constraints force them to be made; the CSP formulation makes it clear, however, that whether or not a decision is forced depends on the deductive power of the underlying constraint engine. Moreover, in the common situation in which no decisions are forced, it is still possible to take a “least commitment” approach by minimizing the degree of commitment of the decision made. The CSP formulation allows these finer distinctions to be made.

5 Related work

Virtually any planner that doesn't simply do a brute-force search of the state space can be viewed as doing at least *some* of its work by posting constraints. Modern planners typically use codesignation constraints, as formalized by Chapman [3], in the process of binding plan step parameters. Another clear example is the use of causal links [10, 12], or their predecessor, protection intervals [21], to constrain the temporal ordering of plan steps. Allen and Koomen [1] and Kambhampati [8] generalize the notion of temporal and causal constraints, respectively.

Planners that make more extensive use of constraints include Zeno [13], O-Plan2 [22], and Collage [9]. Zeno uses constraints and temporal intervals to reason about goals with deadlines and continuous change. O-Plan2 makes it possible for a number of specialized “constraint managers” to work on a plan, all sharing a constraint representation that allows them to interact; resource utilization is handled by one manager, temporal reasoning by another, and so on. One intention behind this design is that as better modules become available for specific kinds of reasoning, they can be incorporated relatively easily into the system. Where O-Plan2, Zeno and Descartes deal primarily with fairly standard, low-level constraints, such as temporal and mathematical constraints, Collage uses high-level constraints that are an integral part of the algorithm itself. For example, task decomposition is accomplished by applying a *decompose* constraint.

Descartes differs from these three planners in its use of active postponement. Zeno, O-Plan2 and Collage all maintain an agenda, representing decisions yet to be made, and to varying extents, take a passive postponement approach in handling that agenda.

Another approach to constraint-based planning is found in MOLGEN [20], a system for planning experiments in molecular genetics. MOLGEN uses constraints on variables to represent certain kinds of goal interactions in a partial plan. In one example it is necessary that the bacterium from one step in an experiment be biologically compatible with the vector from another step; rather than searching for a bacterium and a vector that satisfy this constraint, MOLGEN posts the constraint and delays selecting particular objects until the selection is further constrained by other steps in the experiment.

Contrast this behavior with that of a POCL planner. In setting up the same planning problem for one of those planners we might write the precondition *bio-compat(?b, ?v)* to indicate that the bacterium and vector, represented by the variables *?b* and *?v*, respectively, must be biologically compatible. But that precondition plays a very different role from the constraint posted by MOLGEN. In a POCL planner, while such a precondition remains an open condition it plays no role in restricting the values that can be taken by the two variables; establishing the open condition may result in an early commitment to a particular value. With MOLGEN, on the other hand, once the constraint is posted it serves to rule out impossible values for the two variables, and helps guide the selection of operators to instantiate as interacting steps are added to the plan. The Descartes algorithm may be seen as taking a similar constraint-posting approach, but extending it to apply to all decisions, not just variable binding, and placing it within a more uniform framework. Descartes, however, does not have the hierarchical decomposition used in MOLGEN.

6 Future research

Descartes combines existing research in the areas of temporal world models, constraint satisfaction, and least-commitment planning. The Descartes algorithm also offers significant improvements in the representational ability and efficiency of planning. These improvements are not *ad hoc*, but result from the combination of a clean, uniform plan representation in a more powerful world model and the application of CSP techniques to *all* decisions. In this paper we describe one particular aspect of the Descartes approach, active postponement of plan decisions.

We see future research with Descartes proceeding in several directions. As a framework for the experimental evaluation of decision strategies, Descartes offers an opportunity to characterize and improve our understanding of the value (if any) of early commitments. It may seem intuitively that the least-commitment approach should always be superior, and prior successes with least-commitment planning support that view. If so, then the Descartes framework allows the least-commitment approach to be applied uniformly to all decisions. Early commitments, however, place tighter constraints on variables; tighter constraints, in turn, typically allow CSP reduction techniques to do more work, eliminating impossible values and thus reducing the search. This points to the possibility that early commitments may, under some circumstances, actually improve search efficiency over an approach that postpones every decision as much as possible. Characterizing the tradeoffs involved in this question is one goal of future research.

Another avenue of future research is to leverage innovations in CSP research, using the Descartes framework to apply those techniques to planning problems. For example, some CSP algorithms allow constraints to vary in strength, from constraints that must be satisfied to constraints that only represent some degree of preference [2]. By in-

corporating this CSP technique into the underlying mechanism of Descartes, planning problems could have goals weighted by preference; this in turn could form the basis for an anytime planning algorithm in which successive iterations satisfy those preferences to greater extents.

References

- [1] J. Allen and J. Koonen. Planning using a temporal world model. In *Proc. IJCAI*, pages 741–747, 1983.
- [2] A. Borning, B. Freeman-Benson, and M. Wilson. Constraint hierarchies. *Lisp and symbolic computation*, 5:223–270, 1992.
- [3] D. Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32(3):333–378, 1987.
- [4] K. Currie and A. Tate. O-Plan: the open planning architecture. *Artificial Intelligence*, 52:49–86, 1991.
- [5] D. Joslin. Plan generation as dynamic constraint satisfaction. Forthcoming PhD dissertation from the University of Pittsburgh, 1995.
- [6] D. Joslin and M. E. Pollack. Least-cost flaw repair: A plan refinement strategy for partial-order planning. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI)*, pages 1004–1009, Seattle, WA, 1994.
- [7] S. Kambhampati. Planning as refinement search: A unified framework for comparative analysis of search space size and performance. Technical Report TR-93-004, Arizona State University Department of Computer Science and Engineering, 1993.
- [8] S. Kambhampati. Multi-contributor causal structures for planning: a formalization and evaluation. *Artificial Intelligence*, 69(1-2):235–278, 1994.
- [9] A. L. Lansky. Action-based planning. In *Proceedings of the Second International Conference on AI Planning Systems (AIPS)*, pages 110–115, Chicago, IL, 1994.
- [10] D. McAllester and D. Rosenblitt. Systematic nonlinear planning. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 634–639, Anaheim, CA, 1991.
- [11] S. Minton, J. L. Bresina, and M. Drummond. Commitment strategies in planning: A comparative analysis. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, pages 259–265, 1991.
- [12] J. Penberthy and D. Weld. UCPOP: A sound, complete, partial order planner for ADL. In *Proceedings of the Third International Conference on Knowledge Representation and Reasoning*, pages 103–114, Cambridge, MA, 1992.
- [13] J. S. Penberthy and D. S. Weld. Temporal planning with continuous change. In *Proc. AAAI-94*, pages 1010–1015, 1994.
- [14] M. Peot and D. E. Smith. Threat-removal strategies for partial-order planning. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 492–499, Washington, D.C., 1993.
- [15] M. A. Peot and D. E. Smith. Threat-removal strategies for partial-order planning. In *Proc. AAAI-93*, pages 492–499, 1993.
- [16] M. E. Pollack, D. Joslin, A. Nunes, S. Ur, and E. Ephrati. Experimental investigation of an agent-commitment strategy. Technical Report 94-31, Univ. of Pittsburgh Dept. of Computer Science, Pittsburgh, PA, 1994.
- [17] M. E. Pollack and M. Ringuette. Introducing the Tileworld: Experimentally evaluating agent architectures. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 183–189, Boston, MA, 1990.
- [18] M. E. Pollack, T. Znati, E. Ephrati, D. Joslin, S. Lauzac, A. Nunes, N. Onder, Y. Ronen, and S. Ur. The dipart project: A status report. In *Proceedings of the Annual ARPI Meeting*, Tucson, AZ, 1994.
- [19] L. Schubert and A. Gerevini. Accelerating partial order planners by emphasizing deductive choices. Technical Report TR-570, Dept. of CS, University of Rochester, 1995.
- [20] M. Stefik. Planning with constraints (MOLGEN: Part 1). *Artificial Intelligence*, 16:111–140, 1981.

- [21] A. Tate. Generating project networks. In *Proceedings of IJCAI-77*, pages 888–893, Cambridge, MA, 1977.
- [22] A. Tate, B. Drabble, and J. Dalton. Reasoning with constraints within O-Plan2. Technical Report ARPA-RL/O-Plan2/TP/6 Version 1, Artificial Intelligence Applications Institute, University of Edinburgh, 1994.
- [23] E. Tsang. *Foundations of Constraint Satisfaction*. Academic Press, 1993.