

Is “early commitment” in plan generation ever a good idea?

David Joslin

Computational Intelligence Research Laboratory
1269 University of Oregon
Eugene, OR 97403
joslin@cirl.uoregon.edu

Martha E. Pollack

Department of Computer Science
and Intelligent Systems Program
University of Pittsburgh
Pittsburgh, PA 15260
pollack@cs.pitt.edu

Abstract

Partial-Order Causal Link planners typically take a “least-commitment” approach to some decisions (notably, step ordering), postponing those decisions until constraints force them to be made. However, these planners rely to some degree on early commitments in making other types of decisions, including threat resolution and operator choice. We show why existing planners cannot support full least-commitment decision-making, and present an alternative approach that can. The approach has been implemented in the *Descartes* system, which we describe. We also provide experimental results that demonstrate that a least-commitment approach to planning can be profitably extended beyond what is done in POCL and similar planners, but that taking a least-commitment approach to every planning decision can be inefficient: early commitment in plan generation is sometimes a good idea.

Introduction

The “least-commitment” approach to plan generation has, by and large, been successful where it has been tried. Partial-Order Causal Link (POCL) planners, for example, typically take a least-commitment approach to decisions about step ordering, postponing those decisions until constraints force them to be made. However, these planners rely to some degree on early commitments for other decisions, including threat resolution and choice of an operator to satisfy open conditions. An obvious question is whether the least-commitment approach should be applied to *every* planning decision; in other words, is early commitment ever a good idea?

An obstacle to addressing this question experimentally arises from the way in which POCL (and similar) planners manage decision-making. They take what we call a *passive postponement* approach, choosing one decision at a time to focus on, and keeping all the other, postponed decisions on an “agenda.” Items on the agenda play no role in planning until they are selected for consideration, despite the fact that they may impose constraints on the planning process.

In this paper, we present experimental evidence of the efficiency penalty that can be incurred with passive postponement. We also present an alternative approach, *active postponement*, which 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. We present experimental results indicating that a least-commitment approach to planning can be profitably extended beyond what is done in most planners. We also demonstrate that taking a least-commitment approach to every planning decision can be inefficient: early commitment in plan generation is sometimes a good idea.

Passive postponement

POCL algorithms use *refinement search* (Kambhampati, Knoblock, & Yang 1995). A node N (a partial plan) is selected for refinement, and a flaw F (a threat or open condition) from N is selected for repair. Successor nodes are generated for each of the possible repairs of F . All other (unselected) flaws from the parent node are inherited by these successor nodes. Each flaw represents a decision to be made about how to achieve a goal or resolve a threat; thus, each unselected flaw represents a postponed decision. We term this approach to postponing decisions *passive postponement*. Decisions that are postponed in this manner play no role in planning until they are actually selected.

Passive postponement of planning decisions can incur severe 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: one or more successor nodes will be generated, each inheriting the unrepairable flaw, and each, therefore, also a dead end. In this manner, a single node with a fatal flaw may generate a large number of successor nodes, all dead ends.

The propagation of dead-end nodes is an instance of a more general problem. Similar penalties are paid when a flaw that can be repaired in only one way—a “forced” repair—is delayed; in that case, the forced

repair may have to be repeated in multiple successor nodes. Passive postponement also means that interactions among the constraints imposed by postponed decisions are not recognized until all the relevant decisions have been selected for repair.

The propagation of dead-end nodes is not just a theoretical problem; it can be shown experimentally to cause serious efficiency problems. We ran UCPOP (Penberthy & Weld 1992) on the same test set of 49 problems from 15 domains previously used in (Joslin & Pollack 1994), with the default search heuristics provided by UCPOP. As in the earlier experiments, 32 of these problems were solved by UCPOP, and 17 were not, within a search limit of 8000 nodes generated. We counted the number of nodes examined, and the number of those nodes that were immediate successors of dead-end nodes, i.e., nodes that would never have been generated if fatal flaws were always selected immediately. For some problems, as many as 98% of the nodes examined were successors of dead-end nodes. The average for successful problems was 24%, and for unsuccessful problems, 48%. See (Joslin 1996) for details.

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 (Joslin & Pollack 1994). Indeed, LCFR is a step in the direction of least-commitment planning, because it prefers decisions that are forced to ones that are not. However, even with an LCFR-style strategy, postponed decisions do not play a role in reasoning about the plan until they are selected. Because of this, LCFR can only recognize forced decisions (including dead-ends) that involve just a single flaw on the agenda, considered in isolation. When a decision is forced as a result of flaw interactions, LCFR will not recognize it as forced, and thus may be unable to take a least-commitment approach.

Overview of active postponement

The *active postponement* approach to planning recognizes that flaws represent decisions that will eventually have to be made, and that these postponed decisions impose constraints on the plan being developed. It represents these decisions with constrained variables whose domains represent the available options, and posts constraints that represent correctness criteria on those still-to-be-made decisions. A general-purpose constraint engine can then be used to enforce the constraints throughout the planning process.

To illustrate active postponement for threat resolution, consider a situation in which there is a causal link from some step A to another step B . Assume a third step C , just added to the plan, threatens the causal link from A to B . A constrained variable, D_t , is introduced, representing a decision between promotion and demotion, i.e., $D_t \in \{p, d\}$, and the following con-

straints will be posted:

$$\begin{aligned}(D_t = p) &\rightarrow \text{after}(C, B) \\ (D_t = d) &\rightarrow \text{before}(C, A)\end{aligned}$$

The decision about how to resolve the threat can then be postponed, and made at any time by binding D_t .

Suppose that at some later point it becomes impossible to satisfy $\text{after}(C, B)$. The constraint engine can deduce that $D_t \neq p$, and thus $D_t = d$, and thus C must occur before A . Similarly, if the threat becomes unresolvable, the constraint engine can deduce that $D_t \in \{p, d\}$ cannot be satisfied, and thus that the node is a dead end. This reasoning occurs automatically, by constraint propagation, without the need to “select” the flaw that D_t represents.

When a threat arises, all of the options for resolving that threat are known; other changes to the plan may eliminate options, but cannot introduce new options. Active postponement for goal achievement is more complex, because the repair options may not all be known at the time the goal arises: steps added later to the plan may introduce new options for achieving a goal. To handle this, we allow the decision variables for goal achievement (termed *causal variables*) to have *dynamic domains* to which new values can be added. We notate dynamic domains using ellipses, e.g., $D_g \in \{\dots\}$ represents the decision about how to achieve some goal g , for which there are not yet any candidate establishers. An empty dynamic domain does not indicate a dead end, because new options may be added later; it does, however, mean that the problem cannot be solved unless some new option is introduced. Suppose that at a later stage in planning, some new step F is introduced, with g as an effect. An active postponement planner will expand the domain so that $D_g \in \{F, \dots\}$. Constraints will be posted to represent the conditions, such as parameter bindings, that must be satisfied for F to achieve g .

Least-commitment planning

The active-postponement approach just sketched has been implemented in the Descartes planning system. Descartes provides a framework within which a wide variety of planning strategies can be realized (Joslin 1996). In this section, we provide the details of the algorithm used by Descartes to perform fully “least-commitment” planning: LC-Descartes.

LC-Descartes (Figure 1) can be viewed as performing plan generation *within a single node* that contains both a partial plan Π and a corresponding CSP, C . Π contains a set of steps, some of which have been only tentatively introduced. C contains a set of variables and constraints on those variables. Variables in C represent planning decisions in Π . For example, each precondition p of a step in Π will correspond to a causal variable in C representing the decision of how to achieve p . Parameters of steps in Π will correspond to variables in C representing the binding options.

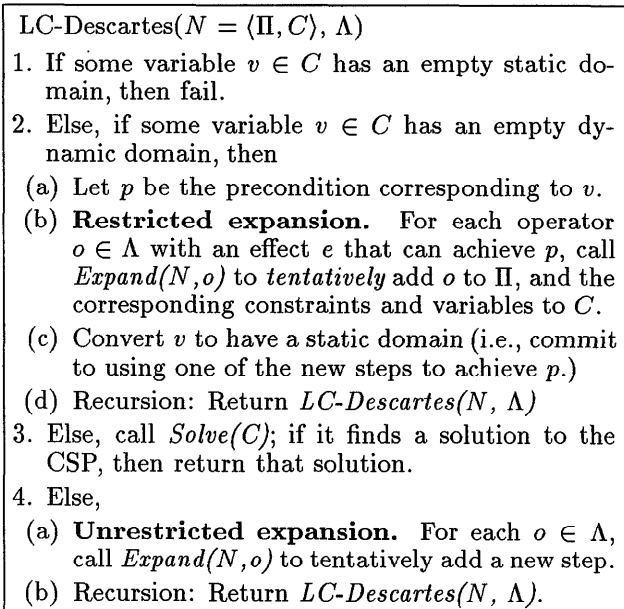


Figure 1: LC-Descartes algorithm

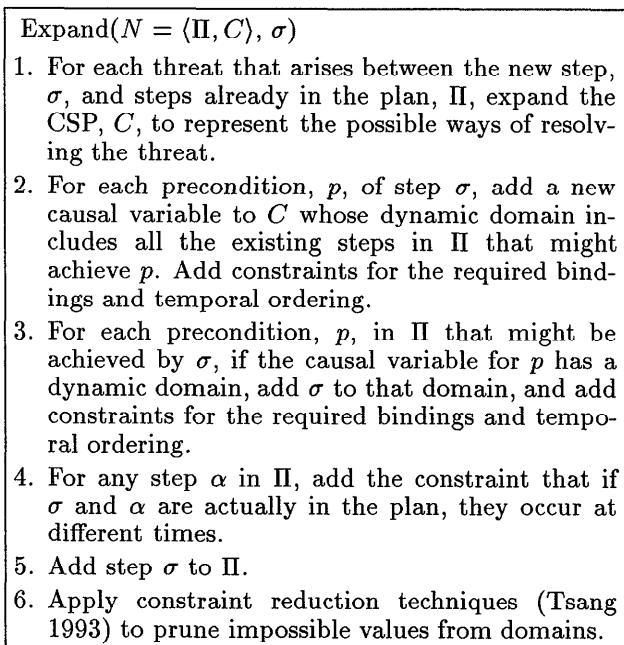


Figure 2: Expand algorithm

Initially, LC-Descartes is called with a node to which only pseudo-steps for the initial and goal state, S_i and S_g , have been added. Following the standard planning technique, S_i 's effects are the initial conditions, and S_g 's preconditions are the goal conditions. The third argument to LC-Descartes is the operator library, Λ .

LC-Descartes first checks whether any variable has an empty static domain, indicating failure of the planning process. If failure has not occurred, it checks

whether any causal variable v has an empty *dynamic* domain, indicating a forced decision to add some step to achieve the goal corresponding to v . In this case, it invokes the *Expand* function (see Figure 2), to add a tentative step for each possible way of achieving the goal, postponing the decision about which might be used in the final plan. As *Expand* adds each new, tentative step to the plan, it also expands the CSP (1) to resolve threats involving the new step, (2) to allow any step already in the plan to achieve preconditions of the new step, (3) to allow the new step to achieve preconditions of steps already in the plan, and (4) to prevent the new step from occurring at the same time as any step already in the plan. This approach to goal achievement generalizes multi-contributor causal structures (Kambhampati 1994).

If no variable has an empty domain, LC-Descartes invokes *Solve*, which applies standard CSP search techniques to try to solve the CSP. At any point, the CSP has a solution if and only the current set of plan steps can be used to solve the planning problem. A solution of the CSP must, of course, bind all of the variables, satisfying all of the constraints. Binding all of the variables means that all planning decisions have been made—the ordering of plan steps, parameter bindings, etc. Satisfying all of the constraints means that these decisions have been made correctly.

In *Solve*, dynamic domains may be treated as static since what we want to know is whether the current set of plan steps are sufficient to solve the planning problem; for this reason, *Solve* uses standard CSP solution techniques. We will assume that *Solve* performs an exhaustive search, but this is not required in practice.

If *Solve* is unsuccessful, then LC-Descartes performs an *unrestricted expansion*, described in the example below. The algorithm continues in this fashion until failure is detected or a solution is found.

A short example. Consider the Sussman Anomaly Blocks World problem, in which the goal is to achieve $on(A, B)$, designated $g1$, and $on(B, C)$, designated $g2$, from an initial state in which $on(C, A)$, $on(A, Table)$, and $on(B, Table)$. The initial CSP is formed by calling *Expand* to add the pseudo-actions for the initial and goal states. At this point, the only dynamic variables will be the causal variables for $g1$ and $g2$; call these D_{g1} and D_{g2} . Since neither goal can be satisfied by any step currently in the plan (i.e., they aren't satisfied in the initial state), both of these variables have empty domains.

There are no empty static domains, but both D_{g1} and D_{g2} have empty dynamic domains. LC-Descartes selects one of them; assume it selects D_{g1} . It performs a *restricted expansion* to add one instance of each action type that might achieve this goal. For this example, suppose we have only one action type, *puton*($?x, ?y, ?z$), that takes block $?x$ off of $?z$ and puts $?x$ on $?y$. The new *puton* step will be restricted so that it

must achieve $on(A, B)$, which here means that the new step is $puton(A, B, ?z1)$; call this step $S1$. Adding this step means that $D_{g1} = S1$. (The domain of D_{g1} is no longer dynamic because a commitment is made to use the new step; this also causes $S1$ to be actually in the plan, not just tentatively.) New variables and constraints will also be added to the CSP for any threats that are introduced by this new step.

The resulting CSP again has a variable with an empty dynamic domain, D_{g2} , corresponding to the goal $on(B, C)$. A restricted expansion adds the step $puton(B, C, ?z2)$, and adds this step ($S2$) to the domain of D_{g2} . Threat resolution constraints and variables are again added. Some of the preconditions of $S1$ are potentially satisfiable by $S2$, and vice versa, and the corresponding variables will have their dynamic domains expanded appropriately.

At this point, the CSP turns out to have no variables with empty domains. This indicates that *as far as can be determined by CSP reduction alone*, it is possible that the problem can be solved with just the current set of plan steps. *Solve* is called, but fails to find a solution; this indicates that, in fact, the current plan steps are not sufficient.

The least-commitment approach demands that we do only what we are forced to do. The failure to find a solution told us that we are forced to expand the plan, adding at least one new step. Unlike the restricted expansions, however, we have no information about which precondition(s) cannot be satisfied, and therefore, do not know which action(s) might be required. That is, there are no empty dynamic domains, so every condition has at least one potential establisher, but conflicts among the constraints prevent establishers being selected for all of them.

Under these conditions, the least-commitment approach requires that we add new steps that can achieve any precondition currently in the plan. In LC-Descartes, this is an *unrestricted expansion*, and is accomplished by adding one step of each type in the operator library. In the current example, this is simplified by the fact that there is only one action type.

The new step, $S3$, will be $puton(?x3, ?y3, ?z3)$, and as before, variables and constraint will be added to allow $S3$ to achieve any goal it is capable of achieving, and to resolve any threats that were introduced by the addition of $S3$. The CSP still has no variables with empty domains, so *Solve* is again called. This time the standard solution to the Sussman anomaly is found.

As promised by its name, LC-Descartes will only commit to decisions that are forced by constraints. However, this least-commitment behavior requires unrestricted expansions, which are potentially very inefficient. They introduce steps that can be used for *any* goal; note that unlike the first two steps, which were constrained to achieve specific goals, step $S3$ above has no bound parameters. Even worse, a new step of each action type must be introduced. In addition, detecting

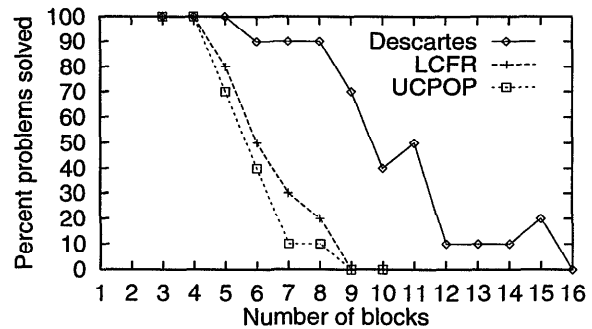


Figure 3: Blocks World, random problems

the need for an unrestricted expansion is much more laborious than checking for variables with empty domains, since it requires that the *Solve* function fail an exhaustive search for a solution.

Experimental results

We begin with a special type of domain, one that has only a single action type. Although unlikely to be of practical interest, such domains allow us to test the limits of LC-Descartes. If unrestricted expansions are a problem even in single-action domains, we know that they will be prohibitive in more realistic domains.

Figure 3 shows experimental results on randomly generated Blocks World problems, using the single-action encoding given in the previous section. Problems were generated using the problem generator provided in the UCPOP 2.0 distribution. The number of blocks was varied from 3 to 16, with 10 problems at each level, for a total of 140 problems. UCPOP, LCFR and LC-Descartes were run on the test problems, with a search limit of 300 CPU seconds on a SPARCstation 20. We report the percentage of problems solved within the CPU limit, and not the number of nodes generated, because LC-Descartes does all its planning within a single node.

UCPOP solved all of the 4-block problems, but its performance fell off rapidly after that point. It dropped below 50 percent at six blocks, and solved no problems of more than eight blocks. LCFR's performance followed a similar pattern. LC-Descartes started to fail on some problems at seven blocks, and dropped below 50 percent at ten blocks. It solves problems of up to fifteen blocks, and fails to solve any problems larger than that. Roughly speaking, on a given set of problems LC-Descartes performed about as well as UCPOP or LCFR performed on problems with half as many blocks. (Doubling the CPU limit did not change the results appreciably.) In interpreting this result, note that the difficulty of blocks world problems increases exponentially with the number of blocks. Also note that because it takes a fully least-commitment approach, in a single-action domain LC-Descartes will always generate minimal-length plans, something not guaranteed by either UCPOP or LCFR.

We investigated what LC-Descartes is doing on the larger problems (see (Joslin 1996) for details), and saw that virtually all of its work is in the form of restricted expansions. Of the 58 successful plans for five-block problems and larger, the average number of steps in each plan was 7.3; of these, only an average of 1.1 steps were added via unrestricted expansions. 31% of these 58 problems were solved with only restricted expansions, another 39% were solved with only one unrestricted expansion. In other words, where LC-Descartes was successful, it was because active postponement allowed it to exploit the structure of the problem enough to either reach a solution directly, or at least get very close to a solution. Success depended on avoiding unrestricted expansions.

EC-Descartes. These results led us to conjecture that the least-commitment approach should be taken at all points *except* those at which LC-Descartes performs unrestricted expansions. We explored this idea by modifying Descartes to make some early commitments: EC-Descartes. EC-Descartes still differs significantly from other planning algorithms, which make early commitments at many points. In POCL planners, for example, threat resolution generates separate successor nodes for promotion and demotion; each node represents a commitment to one step ordering. If both promotion and demotion are viable options, however, then a commitment to either is an early commitment. EC-Descartes avoids this kind of early commitment by posting a disjunctive constraint representing all of the possible options, and postponing the decision about which will be used to resolve the threat.

EC-Descartes and LC-Descartes behave identically except at the point at which LC-Descartes would perform an unrestricted expansion. There, EC-Descartes instead generates more than one successor node. Its objective is to exchange one node that has become under-constrained, and so difficult to solve, for a larger number of nodes that all have at least one variable with an empty domain, static or dynamic. In each of these branches, EC-Descartes then returns to the least-commitment approach, until a solution is found or the problem again becomes under-constrained.

We implemented two versions of EC-Descartes that achieve this objective. The first, EC(1), adopts the simple strategy of selecting the dynamic domain variable with the smallest domain; because only causal variables have dynamic domains, these will be early commitments about action selection. EC(1) generates two successor nodes. In one, all values are removed from the selected variable's domain, forcing a restricted expansion in that node. In the other successor node, the domain is made static. These early commitments are complete; the latter commits to using some step currently in the node, and the former commits to using some step that will be added later. (If EC(1) fails to find a variable with a dynamic domain, it uses EC(2)'s

Problem	LC	EC(1)	EC(2)
1	* (2)	7.2	6.7
2	* (2)	31.5	142.4
3	* (2)	94.7	*
4	8.7 (1)	4.8	5.1
5	38.0 (1)	5.2	9.8
6	* (2)	155.2	*
7	9.0 (1)	7.4	6.2
8	4.0 (0)	3.6	3.6
9	5.8 (1)	5.7	*

CPU times are in seconds; * = exceeded time limit

Figure 4: Early- and least-commitment

strategy instead.)

The second version of EC-Descartes performs early commitments in a manner analogous to the “divide and conquer” technique sometimes used with CSPs. EC(2) performs early commitment at the same time as EC(1), but it selects a *static* variable with minimal domain size, and then generates two successor nodes, each inheriting half of the domain of the selected variable. In that both EC(1) and EC(2) select decisions with minimum domain size (within their respective classes of variables), both bear some resemblance to LCFR.

Figure 4 shows CPU times for LC-Descartes and both versions of EC-Descartes on a set of nine problems from the DIPART transportation planning domain (Pollack *et al.* 1994); it also shows in parentheses the number of unrestricted expansions performed by LC-Descartes. EC(1) solved all of the problems, while EC(2) failed on three problems, and LC-Descartes failed to solve four, hitting a search limit of 600 CPU seconds. On all but the “easiest” problem (# 8), LC-Descartes needs to resort to at least one unrestricted expansion, and it failed on all the problems on which it performed a second unrestricted expansion. Although this domain only has three action types, the added overhead of carrying unneeded (tentative) steps, and all of the associated constraints, is considerable. Not surprisingly (at least in retrospect) the fully least-commitment approach loses its effectiveness rapidly after the transition to unrestricted expansions occurs. The relative advantage of EC(1) over EC(2) suggests that early commitments on action selection are particularly effective.

Related work

Work related to LCFR includes DUnf and DMin (Peot & Smith 1993), branch-1/branch-n (Currie & Tate 1991), and ZLIFO (Schubert & Gerevini 1995). DMin, which enforces ordering consistency for postponed threats, could be viewed as a “weakly active” approach. To a lesser extent, even LCFR and ZLIFO could be thought of as using “weakly active” postponement, since enough reasoning is done about postponed decisions to detect flaws that become dead ends.

Virtually all modern planners do *some* of their work by posting constraints, including codesignation con-

straints on possible bindings, and causal links and temporal constraints on step ordering. Allen and Koomen (Allen & Koomen 1990) and Kambhampati (Kambhampati 1994) generalize the notions of temporal and causal constraints, respectively.

Planners that make more extensive use of constraints include Zeno (Penberthy & Weld 1994) and O-Plan (Tate, Drabble, & Dalton 1994). Zeno uses constraints and temporal intervals to reason about goals with deadlines and continuous change. O-Plan 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. Both Zeno and O-Plan maintain an agenda; Descartes differs from them in its use of active postponement.

Previous work that has used constraints in a more active sense during plan generation includes (Stefik 1981; Kautz & Selman 1992; Yang 1992). MOLGEN (Stefik 1981) posts constraints on variables that represent certain kinds of goal interactions in a partial plan. These constraints then guide the planning process, ruling out choices that would conflict with the constraint. Descartes can 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. Kautz and Selman have shown how to represent a planning problem as a CSP, given an initial user-selected set of plan steps; if a solution cannot be found using some or all of those steps, an expansion would be required, much like an unrestricted expansion in LC-Descartes. WATPLAN (Yang 1992) uses a CSP mechanism to resolve conflict among possible variable bindings or step orderings; its input is a possibly incorrect plan, which it transforms to a correct one if possible. WATPLAN will not extend the CSP if the input plan is incomplete.

Conclusions

The Descartes algorithm transforms planning problems into dynamic CSPs, and makes it possible to take a fully least-commitment approach to plan generation. This research shows that the least-commitment approach can be profitably extended much further than is currently done in POCL (and similar) planners.

There are, however, some fundamental limits to the effectiveness of the least-commitment approach; early commitments are sometimes necessary. In particular, one can recognize that constraints have ceased to be effective in guiding the search for a plan, and at that point shift to making early commitments. These early commitments can be viewed as trading one node whose refinement has become difficult for some larger number of nodes in which constraints force restricted expansions to occur, i.e., trading one "hard" node for several "easy" nodes. One direction for future research will be to look for more effective techniques for making this kind of early commitment.

Acknowledgements. This research has been supported by the Air Force Office of Scientific Research (F49620-91-C-0005), Rome Labs (RL)/ARPA (F30602-93-C-0038 and F30602-95-1-0023), an NSF Young Investigator's Award (IRI-9258392), an NSF CISE Postdoctoral Research award (CDA-9625755) and a Mellon pre-doctoral fellowship.

References

- Allen, J., and Koomen, J. 1990. Planning using a temporal world model. In *Readings in Planning*. Morgan Kaufmann Publishers. 559-565.
- Currie, K., and Tate, A. 1991. O-plan: The open planning architecture. *Art. Int.* 52:49-86.
- Joslin, D., and Pollack, M. E. 1994. Least-cost flaw repair: A plan refinement strategy for partial-order planning. In *Proc. AAAI-94*, 1004-1009.
- Joslin, D. 1996. Passive and active decision postponement in plan generation. Ph.D. dissertation, Intelligent Systems Program, University of Pittsburgh.
- Kambhampati, S.; Knoblock, C. A.; and Yang, Q. 1995. Planning as refinement search: A unified framework for evaluating design tradeoffs in partial-order planning. *Art. Int.* 76(1-2):167-238.
- Kambhampati, S. 1994. Multi-contributor causal structures for planning: a formalization and evaluation. *Art. Int.* 69(1-2):235-278.
- Kautz, H. and Selman, B. 1992. Planning as Satisfiability. *Proc. ECAI-92* Vienna, Austria, 1992, 359-363.
- Penberthy, J. S., and Weld, D. 1992. UCPOP: A sound, complete, partial order planner for ADL. In *Proc. 3rd Int. Conf. on KR and Reasoning*, 103-114.
- Penberthy, J. S., and Weld, D. 1994. Temporal planning with continuous change. In *Proc. AAAI-94*, 1010-1015.
- Peot, M., and Smith, D. E. 1993. Threat-removal strategies for partial-order planning. In *Proc. AAAI-93*, 492-499.
- Pollack, M. E.; Znati, T.; Ephrati, E.; Joslin, D.; Lauzac, S.; Nunes, A.; Onder, N.; Ronen, Y.; and Ur, S. 1994. The DIPART project: A status report. In *Proceedings of the Annual ARPI Meeting*.
- Schubert, L., and Gerevini, A. 1995. Accelerating partial order planners by improving plan and goal choices. Tech. Rpt. 96-607, Univ. of Rochester Dept. of Computer Science.
- Stefik, M. 1981. Planning with constraints. *Art. Int.* 16:111-140.
- Tate, A.; Drabble, B.; and Dalton, J. 1994. Reasoning with constraints within O-Plan2. Tech. Rpt. ARPA-RL/O-Plan2/TP/6 V. 1, AIAI, Edinburgh.
- Tsang, E. 1993. *Foundations of Constraint Satisfaction*. Academic Press.
- Yang, Q. 1992. A theory of conflict resolution in planning. *Art. Int.* 58(1-3):361-392.