# Analogical Replay for Efficient Conditional Planning*

**Jim Blythe** **Manuela Veloso**
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213, USA
jblythe,veloso@cs.cmu.edu

### Abstract

Recently, several planners have been designed that can create conditionally branching plans to solve problems which involve uncertainty. These planners represent an important step in broadening the applicability of AI planning techniques, but they typically must search a larger space than non-branching planners, since they must produce valid plans for each branch considered. In the worst case this can produce an exponential increase in the complexity of planning. If conditional planners are to become usable in real-world domains, this complexity must be controlled by sharing planning effort among branches. Analogical plan reuse should play a fundamental role in this process. We have implemented a conditional probabilistic planner that uses analogical plan replay to derive the maximum benefit from previously solved branches of the plan. This approach provides valuable guidance for when and how to merge different branches of the plan and exploits the high similarity between the different branches in a conditional plan, which have the same goal and typically a very similar state. We present experimental data in which analogical plan replay significantly reduces the complexity of conditional planning. Analogical replay can be applied to a variety of conditional planners, complementing the plan sharing that they may perform naturally.

## Introduction

Most AI research in planning systems has been made under the assumption that the planning domain is certain: the planner's initial state is known absolutely, the effects of all actions are perfectly predictable and the planner is the only agent acting in an otherwise unchanging world. These assumptions have allowed sound fundamental work to be done and many interesting planning systems to be built. However they have also limited the applicability of these systems for real problems, where the assumptions rarely hold.

Recently several planners have been designed that relax these assumptions. They fall into two broad groups: those that extend techniques for solving Markov decision processes (MDPs) such as policy iteration (Dean & Lin 1995), and those that extend AI planning algorithms such as SNLP (Draper, Hanks, & Weld 1994). As remarked by Boutilier et al (Boutilier, Dean, & Hanks 1995), the two groups of planners differ only in the emphasis of available techniques and do not make different assumptions.

In this paper we concentrate on the efficiency problem for systems based on classical AI planners. Many useful techniques have been developed to improve the efficiency of classical planners, and we are interested in the extent to which some of them can be used in planners for domains with uncertainty. In particular in this work we investigate the use of derivational analogy in conditional planners. The essence of the method is to flexibly reuse the planning experience across the conditional branches, thus avoiding the need for unnecessary repeated search effort. The use of derivational analogy in this context is similar to internal analogy (Hickman, Shell, & Carbonell 1990) in which repeated solutions to subproblems are reused within the same problem.

We have implemented a conditional probabilistic planner that extends Prodigy 4.0 (Veloso *et al.* 1995). We then integrated it with analogical plan replay to derive the maximum benefit from previously solved branches of the plan. This approach provides valuable guidance for when and how to merge different branches of the plan and exploits the similarity that can exist between different branches in a conditional plan, which have the same goal and typically a very similar state. We present experimental data in which analogical plan replay significantly reduces the complexity of conditional planning. Finally we discuss some general issues in plan sharing and contrast our approach with those of other systems.

## Conditional Planning

Consider a simple planning problem in which a package is to be loaded into a truck. In the initial state, the package is at the depot and the truck is at the warehouse. However, consider also that, in the time it takes to drive the truck to the depot, the package can be misplaced with probability 0.5. When this happens, the package is transferred from the depot to the lost-property department, from where it cannot be

 This research is sponsored as part of the DARPA/RL Knowledge Based Planning and Scheduling Initiative under grant number F30602-95-1-0018.

lost. The following branching plan solves this problem: drive the truck to the depot and if the package is still there load it into the truck, otherwise drive to lost property and load the package into the truck.

Several planning systems are capable of solving problems like this, such as CNLP (Peot & Smith 1992), Cassandra (Pryor & Collins 1993) and C-Buridan (Draper, Hanks, & Weld 1994). We present here a conditional planner that is an extension to PRODIGY4.0((Veloso *et al.* 1995)), which we use to motivate and explore the use of analogy in conditional planners.

Our conditional planner, B-PRODIGY (for *branching* Prodigy), is implemented within the Weaver architecture for planning under uncertainty. Weaver is a probabilistic planner taking as input a probability distribution of initial world states, a set of operators with probability distributions of context-dependent effects and a set of probabilistic external events, which are triggered by the state, independently of the actions taken by the planner. Weaver builds a series of plans with increasing probabilities of success and iteratively calls B-PRODIGY to find improved versions of the plan (Blythe 1994). At each iteration, Weaver decides which sources of uncertainty to account for and which to ignore. It hides the latter from B-PRODIGY, and allows B-PRODIGY to reason about the relevant external events by representing them as possible effects of actions. Like Cassandra, B-PRODIGY then searches for a conditional plan that will succeed in every eventuality that it is aware of.

B-PRODIGY's search space is similar to that of PRODIGY4.0, and the search control modules that can be applied to PRODIGY4.0 can also be applied to B-PRODIGY with a small degree of modification. These modules provide suggestions to PRODIGY4.0 in its search, usually through control rules. In particular the analogical replay method, implemented in Prodigy/Analogy, uses previously successful problem-solving traces to guide Prodigy.

The remainder of this paper concentrates on B-PRODIGY as a separate conditional planner that does not deal with probabilities, in order to cover the use of analogical replay in this context in detail.

## B-PRODIGY

PRODIGY4.0 represents a partial plan in two parts, a *head plan*, which is a totally ordered set of steps applicable from the initial state, and a *tail plan*, which is a partially ordered set of steps in which each step is linked to the step whose precondition it is used to achieve. If there is a link from step $A$ to step $B$ we say that $A$ is an *establisher* of $B$. The head plan determines a unique *current state*, the result of applying the head plan to the initial state. PRODIGY4.0 begins with an empty head plan and a tail plan consisting of a step representing the goal statement, as the root of the partial order. The algorithm terminates when the goal is satisfied in the current state, and then the head plan represents a valid plan.

B-PRODIGY has two main additions to PRODIGY4.0. First, steps in both the head plan and the tail plan are associated with *contexts*, which represent the branches of the plan in which the step is proposed to be used. Contexts are introduced when a branching action is added to the tail plan, and correspond to the different possible outcomes of the action. A step may belong to several contexts, which must be a subset of those of the step it is linked to. Second, the head plan is no longer a totally-ordered sequence determining a single current state, but is a branching sequence determining a set of possible states. Multiple recursive calls are made to the B-PRODIGY routine when a branching step is moved from the tail plan to the head plan, one call for each possible result (context) of the step. In each call, the steps in the tail plan that do not match the context are removed, so that the call corresponds to solving the specific branch of the plan for which it is chosen. The several calls together produce a branching head plan. These two alterations are sufficient to create branching plans in uncertain domains. Table 1 shows the algorithm (bold font shows the modification to the original PRODIGY4.0 algorithm).

B-PRODIGY

1. If the goal statement $G$ is satisfied in the current state $C$, then return *Head-Plan*.
2. Either
   - Add an operator $O$ to the *Tail-Plan* to establish some operator $E$. **(The contexts of $O$ must be a subset of the contexts of $E$.)** Or
   - Choose an operator $A$ from *Tail-Plan* whose preconditions are satisfied in the current state, move it to *Head-Plan*, and update the current state. **If $A$ has multiple branches, for each one of them, create a new partial plan and remove from *Tail-Plan* all operators that do not match the appropriate context (branch).**
3. Recursively call B-PRODIGY on each resulting partial plan.

Table 1: Algorithm for B-PRODIGY, based on PRODIGY4.0. Steps which appear only in B-PRODIGY are shown in bold.

If no branching steps are introduced, this algorithm is identical to PRODIGY4.0. Like PRODIGY4.0, it is easy to see that the algorithm is sound, yielding only correct plans.

Figure 1 shows a tail plan with four steps that may be constructed by the partial-order back-chaining algorithm to solve the problem described at the beginning of this section. The truck must be made ready, with step 1 "start-truck" before it can be driven, and the final goal requires that the truck is put away, with step 4 "stop-truck." The arc from "drive to depot" to

"load package at depot" indicates that the former step is an establisher of the latter. We have omitted the preconditions from the diagram.

Step 2 "drive to depot," is a context-producing step that has two possible sets of effects. Contexts $\alpha$ and $\beta$ correspond respectively to the situations where the package is still at the depot and where it is in lost property, when the truck arrives. A step is marked as context-producing externally to B-PRODIGY (in fact by Weaver as we described).

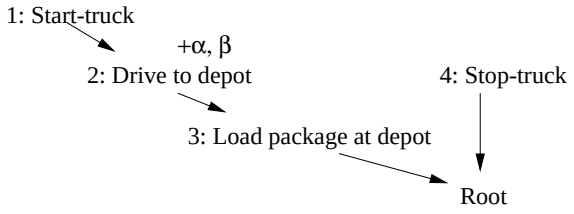


Figure 1: Initial tail plan to solve example problem. The directed arcs are causal links showing that a step establishes a necessary precondition of the step it points to.

When a branching step is introduced into the tail plan, producing new contexts, the other steps are initially assumed to belong to all contexts. This is true of step 3, "load package at depot." However each step's contexts can be restricted to a subset, and new steps can be added to achieve the same goal in the other contexts. In this example, new operators could be introduced both for the top-level goal and the goal for the truck to be at the depot. The branching step is always introduced with a commitment that one of its outcomes will be used to achieve its goal, and all the ancestors of the step in the tail plan must always apply to that context. In this example step 2 was introduced to establish step 3 using context $\alpha$, so step 3 may not be restricted to context $\beta$.

In Figure 2, step 3 has been restricted to context $\alpha$, new steps have been added to achieve the top-level goal in context $\beta$, and steps 1 and 2 been moved to the head plan. Two recursive calls to B-PRODIGY will now be made, one for each context, in each of which B-PRODIGY will proceed to create a totally-ordered (but possibly nonlinear) plan. In context $\alpha$, the steps labelled with $\beta$, driving to and loading at lost-property, will be removed from the tail plan. In context $\beta$, the step labelled with $\alpha$, "load package at depot," will be removed. Step 4, stop-truck, has not been restricted and remains in the tail plan in both contexts. No steps are removed from the head plan, whatever their context. Each recursive call produces a valid totally-ordered plan, and the result is a valid conditional plan that branches on the context produced by the step "drive to depot."

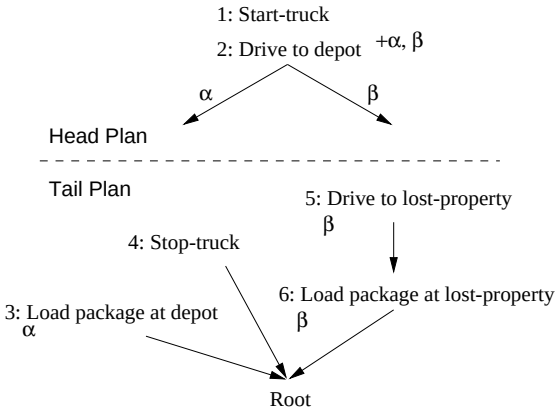


Figure 2: A second stage of the planner. There are now two possible current states for contexts $\alpha$ and $\beta$.

## Sharing planning effort in B-PRODIGY

The ability to create conditional plans is vital to planners that deal with uncertainty, however creating them can lead to high computational overheads because of the need for separate planning effort supporting each of the branches of the conditional plan, measured in terms of the computation required to search for an validate the plan. This problem can be alleviated by sharing as much of the planning effort as possible between different branches. Analogical replay enables sharing the effort to construct plans, while revalidating decisions for each new branch. As the following three examples show, in some cases the planning architecture directly supports sharing this effort and in others it does not.

1. Step 1, start-truck, is shared through the *head plan*. The step is useful for both branches, but is only planned for in one since its effect is shared through the current state. Although in this plan there is only one step, in general an arbitrary amount of planning effort could be shared this way.
2. Step 4, stop-truck, is shared through the *tail plan*. As shown in Figure 2, this step does not have a context label and achieves its goal in either context. Thus when the branching step is moved to the head plan, it remains in the tail plan in each recursive call made to B-PRODIGY, making the planning effort available in each call.
3. Sometimes duplicated planning effort is not architecturally shared as in the last two examples. Suppose that extra set-up steps are required for loading a truck. These would be added to the tail-plan in Figure 2 in two different places, as establishers of steps 3 and 6, and restricted to different contexts in the two places. Thus, the planning effort cannot be shared by the tail-plan unless it is modified from a tree to a DAG structure. But this approach would lead to problems if descendant establishing steps were to depend on the context, for instance on

the location of the truck. We show below how the use of analogy can transfer planning effort of this kind between contexts. Analogy provides an elegant way to handle other kinds of shared steps as well.

## Analogical Replay in B-PRODIGY

Analogical reasoning has been combined with classical operator-based planning in Prodigy as a method to learn to improve planning efficiency (Veloso 1994). This integrated system, Prodigy/Analogy, has been reimplemented in PRODIGY4.0. Its replay functionality makes it suitable to be combined with B-PRODIGY. B-PRODIGY controlled by Weaver is integrated with Prodigy/Analogy. First, a previously visited branch is selected to guide planning for the new branch. By default, the branch that was solved first is used. Next B-PRODIGY is initialized to plan from the branch-point, the point where the new branch diverges from the guiding branch. Then B-PRODIGY plans for the new branch, guided by Prodigy/Analogy, proceeding as usual by analogical-replay: previous decisions are followed if valid, unnecessary steps are not used, and new steps are added when needed. Prodigy/Analogy successfully guides the new planning process based on the high global similarity between the current and past situations. This is particularly well suited for the analogical replay guidance and typically leads to minor interactions and a major sharing of the past planning effort. The smoothness of this integration is made possible by the common underlying framework of the Prodigy planning and learning system.

In this integration of conditional planning and analogy, the analogical replay within the context of different branches of the same problem can be viewed as an instance of internal analogy (Hickman, Shell, & Carbonell 1990). The accumulation of a library of cases is not required, and there is no need to analyze the similarity between a new problem and a potentially large number of cases. The branches of the problem need only to be cached in memory and most of the domain objects do not need to be mapped into new objects, as the context remains the same. While we currently use this policy in our integration, the full analogical reasoning paradigm leaves us with the freedom to reuse branches across different problems in the same domain. We may also need to merge different branches in a new situation.

Table 2 presents the analogical reasoning procedure combined with B-PRODIGY. We follow a single case corresponding to the plan for the last branch visited according to the order selected by Weaver.

The adaptation in the replay procedure involves a validation of the steps proposed by the case. There may be a need to diverge from the proposed case step, because new goals exist in the current branch (step 9). Some steps in the old branch can be skipped, as they may be already true in the new branching situation (step 13). Steps 8 and 12 account for the sharing between different branches and for most of the new planning, since typically the state is only slightly different and most of the goals are the same across branches. This selective use of replay controls the combinatorics of conditional planning.

**procedure** *b-prodigy-analogical-replay*
1. Let $C$ be the guiding case,
   and $C_i$ be the guiding step in the case.
2. Set the initial case step $C_0$ based on the branch point.
3. Let $i = 0$.
4. Terminate if the goal state is reached.
5. Check which type of decision is $C_i$:
6. If $C_i$ adds a step $O$ to the head plan,
7. If $O$ can be added to the current head plan and no tail planning is needed before,
8. then Replay $C_i$; Link new step to $C_i$; goto 14.
9. else Hold the case and call B-PRODIGY, if planning for new goals is needed; goto 5.
10. If $C_i$ adds a step $O_g$ to the tail plan, to achieve goal $g$,
11. If the step $O_g$ is valid and $g$ is needed,
12. then Replay $C_i$; Link new step to $C_i$; goto 15.
13. else Mark unusable all steps dependent on $C_i$;
14. Advance the case to the next usable step $C_j$;
15. $i \leftarrow j$; goto 5.

Table 2: Overview of the analogical replay procedure combined with B-PRODIGY.

## Analysis and Experiments

Each segment of a conditional plan has a corresponding planning cost. If a segment is repeated $k$ times and can be shared but is not, the planner incurs a penalty of $k-1$ times the cost of the segment. Suppose that a plan contains $n$ binary branches in sequence, all of which share steps. Either the first or the last part of the plan may be created $2^n$ times, but with step sharing it may only need to be created once. This exponential cost increase can quickly become a dominant factor in creating plans for uncertain domains.

We have successfully applied our approach to a large realistic oil-spill clean-up domain and continue to apply it to real-world domains. To isolate problem features that are pertinent to performance, a family of synthetic domains was created in order to comprehensively verify experimentally the effect of analogy in conditional planning. These domains allow precise control over the number of branches in a plan, the amount of planning effort that may be shared between branches and the amount that belongs only to each branch.

Table 3 describes the operators. The top-level goal is always g, achieved by the operator Top. There is a single branching operator, Branch, with $N$ different branches corresponding to $N$ contexts. A plan consists of three main segments. The first segment consists of the steps taken before the branch point. These are all in aid of the goal bb, the only goal initially unsatis-

fied, which is achieved by the step Branch. All of the branches of Branch achieve bb, but delete cx and sh. Each branch also adds a unique "context" fact, $c_i$. After the branch point, the second segment is a group of steps unique to each individual branch, in aid of the goal cx. We name each of these segments "$C_i$." Finally, the third "shared" segment contains the steps which are the same in every branch in aid of the goal sh. They must be taken after the branching point, since Branch deletes sh. The planning work done in each segment is controlled by the iterative steps that achieve the predicates iter-bb$_0$, iter-cx$_{i,0}$ and iter-sh$_0$. The plan to achieve iter-sh$_0$, for example, has a length determined by some number $z$ for which iter-sh$_z$ is the initial state. The planner selects the operators S-sh$_k$ which succeed or F-sh$_k$ ($k = 0, \ldots, z$) which fail as their preconditions u-sh$_k$ of the operators A-sh$_k$ are all unachievable. The domain can have $N$ copies of these operators. So the planning effort to solve iter-sh$_0$ is up to $2N \times z$ operators added to the tail plan, all but $z$ of which are removed.

| *Operator* | *Preconds* | *Adds* | *Deletes* |
|---|---|---|---|
| Top | bb, cx, sh | g | – |
| Branch | iter-bb$_0$ | bb,c$_i$ | sh, cx |
| *where $i$ refers to each branch $i$, $i = 1, \ldots, B$* | | | |
| Contx$_i$ | iter-cx$_{i,0}$, c$_i$ | cx | – |
| Shared | iter-sh$_0$ | sh | – |
| F-bb$_l$ | a-bb$_l$ | iter-bb$_l$ | – |
| A-bb$_l$ | u-bb$_l$ | a-bb$_l$ | – |
| S-bb$_l$ | iter-bb$_{l+1}$ | iter-bb$_l$ | – |
| F-cx$_{i,m}$ | a-cx$_{i,m}$ | iter-cx$_{i,m}$ | – |
| A-cx$_{i,m}$ | u-cx$_{i,m}$ | a-cx$_{i,m}$ | – |
| S-cx$_{i,m}$ | iter-cx$_{i,m+1}$ | iter-cx$_{i,m}$ | – |
| F-sh$_k$ | a-sh$_k$ | iter-sh$_k$ | – |
| A-sh$_k$ | u-sh$_k$ | a-sh$_k$ | – |
| S-sh$_k$ | iter-sh$_{k+1}$ | iter-sh$_k$ | – |
| *where $l, k, m$ capture the lengths of the segments* | | | |

Table 3: Operator schemas in the test domain.

Figure 3 shows an example of a plan generated for for a problem with goal g, and initial state cx, sh, iter-bb$_x$, iter-cx$_{1,y}$, ..., iter-cx$_{B,y}$, and iter-sh$_z$.

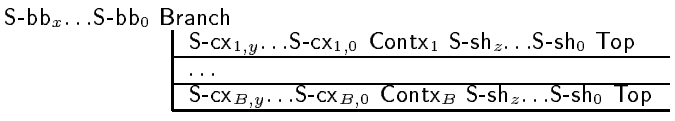


Figure 3: A typical plan in the test domain.

We have performed extensive experiments with a variety of setups. As an illustrative performance graph, Figure 4 shows the planning time in seconds when the number of branches is increased from 1 to 10. For this graph, the final plan has one step in each of the "$C_i$" and the "shared" segments. The domain was chosen so that B-PRODIGY examined 4 search nodes to create a "$C_i$" segment and 96 for the "shared" segment. With less extreme proportions of shared planning time to unique planning time, the shape of the graph is roughly the same and analogical replay still produces significant speedups. With 24 search nodes examined for each unique plan segment, and 72 for the shared segment, B-PRODIGY completes the plan with 10 branches more than twice as quickly with analogical replay as without it.

The improvement in time is similar when the depth and breadth of search are increased for the shared segment and the number of branches is held constant.

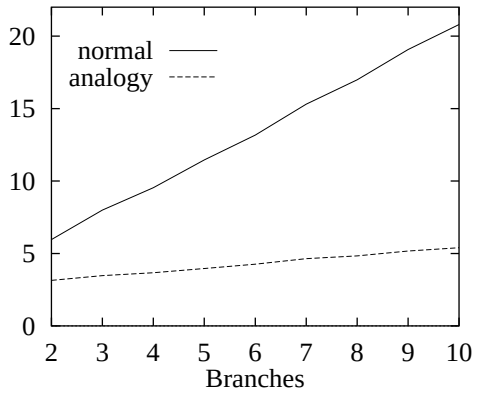


Figure 4: Time in seconds to solve planning problem with and without analogy plotted against the number of branches in the conditional plan. Each point is an average of five planning episodes.

The use of analogical replay in B-PRODIGY is a heuristic based on the assumption that a significant proportion of planning work can be shared between the branches. We tested the limits of this assumption by experiments holding constant both the number of branches and the effort to create the shared segment, and increasing the effort to create each unique segment. Under these conditions, the time taken by B-PRODIGY grows at the same rate whether or not analogical replay is used, because the overhead of replay is small relative to planning effort, and appears constant. When the planning effort for each C-i was increased from 4 to 100 search nodes while the effort to create a shared branch was held constant at 24 search nodes, B-PRODIGY took about 1 second longer with analogy than without it, an overhead of less than 10 per cent when each C-i took 100 search nodes.

## Related Work

In (Peot & Smith 1992), Peot and Smith introduce a planner called CNLP with a representation for partial-order branching plans based on SNLP (McAllester &

Rosenblitt 1991). This representation uses contexts, and has been adopted in B-PRODIGY as well as Cassandra (Pryor & Collins 1993) and C-Buridan (Draper, Hanks, & Weld 1994). All three planners keep track of the branches that steps belong to using *context propagation*, assigning contexts to steps based on those they are causally linked to. C-Buridan's version consists of a "forwards sweep" in which the context of each step is restricted to the disjunction of the contexts of the steps that help establish it, followed by a "backwards sweep" in which the context of each step is restricted to the disjunction of the contexts of the steps it establishes.

C-Buridan combines context propagation with SNLP's ability to use steps already existing in its plan as establishers to produce a simple and elegant technique for sharing steps between different branches of a conditional plan, which are not architecturally shareable in our planner. In solving the example problem in this paper, C-Buridan might begin with a similar candidate plan to B-PRODIGY's tail plan shown in Figure 2. If the step "open-truck" was required to enable loading the truck in steps 3 and 6, C-Buridan could use one step to establish both of them, and share the planning effort to establish that step.

Although elegant, sharing plan steps with context propagation and linking to existing steps is not always as efficient as using analogical replay. Analogical replay can skip parts of an old plan and add new steps while replaying, creating plans that branch and remerge several times if necessary guided by the current state. Context propagation in C-Buridan may result in new context values being propagated over long plans as each new branch is added. Consider the following set of operators, in which the step Branch-op has $n$ branches:

(step Branch-op
(branch-0 adds { g y })
(branch-$i$ adds { q$n$, g, x$i$ } ))
(step do-Q$i$ preconds { q[$i-1$] }
if ( y ) adds { q$i$ } else deletes { g } )

Given the conjunction of g and q$n$ as a goal and q0 initially true, a successful plan uses Branch-op to achieve both goals in every branch except branch-0, where the sub-plan do-Q0, do-Q1, ..., do-Q$n$ is used to achieve q$n$. These operators must not be applied in any other branch since they will delete g. C-Buridan will have to resolve the threat from each step do-Q$i$ in the plan to each branch of Branch-op where it shouldn't be used, $O(n^2)$ threats in all. Analogical replay will only add the extra steps in the branch where they are needed and will take linear time.

On the other hand our implementation of analogy requires one branch to be completed before it can be used to guide others, a restriction that reduces the chance to plan in two branches synergistically. It is also guided by the current state as a heuristic, which can lead to extra work. However, an attractive feature of analogical replay is that it is applicable to all the conditional planners we have mentioned, and can implement several types of plan sharing independently of those explicitly shared by the particular architecture.

## Conclusion

We have shown that analogy can be used to reduce the overhead of producing plans that have many branches, each covering somewhat different situations. This is an example of how machine learning techniques designed for planners that made the assumption of complete information can be brought to bear on planners that relax this assumption. Although different architectural constraints in planners allow different parts of branching plans to be shared easily, analogy provides a general mechanism that can share all types of planning work. The approach is also applicable to partial-order conditional planners such as Cassandra and C-Buridan.

## References


Blythe, J. 1994. Planning with external events. In de Mantaras, R. L., and Poole, D., eds., *Proc. Tenth Conference on Uncertainty in Artificial Intelligence*, 94–101. Morgan Kaufmann.

Boutilier, C.; Dean, T.; and Hanks, S. 1995. Planning under uncertainty: structural assumptions and computational leverage. In *Proc. European Workshop on Planning*. Assissi, Italy: IOS Press.

Dean, T., and Lin, S.-H. 1995. Decomposition techniques for planning in stochastic domains. In *Proc. 14th International Joint Conference on Artificial Intelligence*, 1121 – 1127. Morgan Kaufmann.

Draper, D.; Hanks, S.; and Weld, D. 1994. Probabilistic planning with information gathering and contingent execution. In Hammond, K., ed., *Proc. Second International Conference on Artificial Intelligence Planning Systems*, 31–37. AAAI Press.

Hickman, A. K.; Shell, P.; and Carbonell, J. G. 1990. Internal analogy: Reducing search during problem solving. In Copetas, C., ed., *The Computer Science Research Review*. Carnegie Mellon University.

McAllester, D., and Rosenblitt, D. 1991. Systematic nonlinear planning. In *Proc. Ninth National Conference on Artificial Intelligence*, 634–639. AAAI Press.

Peot, M. A., and Smith, D. E. 1992. Conditional nonlinear planning. In Hendler, J., ed., *Proc. First International Conference on Artificial Intelligence Planning Systems*, 189–197. Morgan Kaufmann.

Pryor, L., and Collins, G. 1993. Cassandra: Planning for contingencies. Technical Report 41, The Institute for the Learning Sciences.

Veloso, M.; Carbonell, J.; Pérez, A.; Borrajo, D.; Fink, E.; and Blythe, J. 1995. Integrating planning and learning: The prodigy architecture. *Journal of Experimental and Theoretical AI* 7:81–120.

Veloso, M. M. 1994. *Planning and Learning by Analogical Reasoning*. Springer Verlag.