Using Classical Planners to Solve Nondeterministic Planning Problems

Ugur Kuter Dana Nau Elnatan Reisner
Department of Computer Science,
Institute for Systems Research, and
Institute for Advanced Computer Studies
University of Maryland, College Park, MD, USA
email: {ukuter,nau,elnatan}@cs.umd.edu

Robert P. Goldman
Smart Information Flow Technologies
(d/b/a SIFT, LLC)
211 N. First St., Suite 300
Minneapolis, MN 55401, USA
email: rpgoldman@SIFT.info

Abstract

Researchers have developed a huge number of algorithms to solve classical planning problems. We provide a way to use these algorithms, unmodified, to generate strong-cyclic solutions in fully-observable nondeterministic planning domains. Our experiments show that when using our technique with FF and SGPlan (two well-known classical planners), its performance compares quite favorably to that of MBP, one of the best-known planners for nondeterministic planning problems.

Introduction

Despite the promising performance of planning algorithms such as MBP (Cimatti et al. 2003), nondeterministic planning problems,[1] in which each action may nondeterministically produce any of several outcomes, are generally very difficult to solve. This difficulty persists even when the planning problems are fully observable, i.e., the states of the world can be completely observed at runtime. One reason is that in nondeterministic planning problems, the planning algorithm must reason about all possible different execution paths to find a plan that works despite the nondeterminism, hence the size of the generated conditional plan may grow exponentially.

In contrast, researchers have developed many highly efficient planning algorithms for domain-independent classical planning. In this paper we describe a way to use these algorithms to solve nondeterministic planning problems. We provide the following contributions:

- NDP, an algorithm that can use any classical planner to generate cyclic solutions in nondeterministic planning domains. NDP accomplishes this without requiring any internal modifications to the classical planner itself. Given a classical planner R and a nondeterministic planning problem P, NDP calls R on a sequence of classical planning problems and uses the results of these calls to construct a solution for P.
- *Conjunctive abstraction*, in which conjuncts of literals are used to represent classes of states rather than individual states. This compresses the state space in a similar way to the Binary Decision Diagrams (BDDs) used in the MBP planner (Cimatti et al. 2003). The compression is less powerful than BDDs, but has the advantage that it can be used in any planner that uses nondeterministic STRIPS operators, without any modifications to the planner.

To provide insight about the strengths and weakness of our approach, we report on experimental tests of NDP and conjunctive abstraction, using four well-known classical planners in three different planning domains:

- NDP worked especially well with FF (Hoffmann and Nebel 2001) and SGPlan (Hsu et al. 2006). In two of our experimental domains (Robot Navigation (Cimatti et al. 2003) and nondeterministic Blocks World (Kuter and Nau 2004)), NDP with those planners dramatically outperformed MBP.
- In our third experimental domain, Hunter-Prey (Kuter et al. 2005), conjunctive abstraction did not provide as much compression as MBP's BDDs did. Consequently, MBP outperformed NDP in this domain regardless of which classical planner we used.

Definitions and Notation

Nondeterministic Planning Domains. Intuitively, a *nondeterministic planning domain* is one in which each action may have more than one possible outcome. Formally, it is a triple $D = (\mathcal{S}, \mathcal{A}, \gamma)$, where $\mathcal{S}$ is a finite set of *states*, $\mathcal{A}$ is a finite set of *actions*, and $\gamma : \mathcal{S} \times \mathcal{A} \to 2^{\mathcal{S}}$ is the *state-transition function*. An action a is *applicable* in s if $\gamma(s, a)$ is nonempty. We let $A_D(s)$ be the set of all actions that are applicable in s in the domain D. If $a \in A_D(s)$, then executing a in s may produce any of the states in $\gamma(s, a)$.

A *policy* is a function from a set $S_\pi \subseteq S$ into $\mathcal{A}$, such that for each $s \in S_\pi$, the action $\pi(s)$ is applicable to s. π's *execution structure* is a digraph Σ_π representing all possible executions of π. Formally, $\Sigma_\pi = (V_\pi, E_\pi)$, where $V_\pi = S_\pi \cup \bigcup\{\gamma(s, \pi(s)) \mid s \in S_\pi\}$ and $E_\pi = \{(s, s') \mid s \in S_\pi, s' \in \gamma(s, \pi(s))\}$. If there is a path in Σ_π from s to s' then s is a π-*ancestor* of s' and s' is a π-*descendant* of s. If the path has length 1, s is a π-*parent* of s' and s' is a π-*child* of s. A node is a *leaf* if it has no π-children.

[1] Unfortunately, the term "nondeterministic planning problem" seems to have two meanings in the current literature: some researchers attach probabilities to the action outcomes (as in a Markov Decision Process), and others omit the probabilities (as in a nondeterministic automaton). Our usage is the latter one.

If S is any set of states, then π's *execution structure on* S, $\Sigma_\pi(S)$, is the subgraph of Σ_π rooted at S. (Note that if π is empty then $\Sigma_\pi(S) = (S, \emptyset)$).

A nondeterministic planning problem is a triple $P = (D, S_0, G)$, where $D = (\mathcal{S}, \mathcal{A}, \gamma)$ is a nondeterministic planning domain, $S_0 \subseteq \mathcal{S}$ is a set of initial states, and $G \subseteq \mathcal{S}$ is a set of *goal states*. A *strong-cyclic solution* for P is a policy π such that every state s in Σ_π is a π-ancestor of at least one goal state and the leaf nodes in Σ_π are goal states (Cimatti et al. 2003). Note that for each initial state in Σ_π, there is at least one path that ends in a goal state and the collection of those paths is called a *weak solution* to P (Cimatti et al. 2003). (Cimatti et al. 2003) also defines *strong* solutions, but we will not need that definition in this paper.

Classical Planning Domains. An action a is *classical* if $|\gamma(s, a)| \leq 1 \; \forall s$, i.e., a never has more than one outcome. A planning domain $D = (\mathcal{S}, \mathcal{A}, \gamma)$ is classical if every action in $\mathcal{A}$ is classical. A planning problem $P = (D, S_0, G)$ is classical if D is classical and there is just one initial state, i.e., $S_0 = \{s_0\}$ for some $s_0 \in \mathcal{S}$.

In a classical planning problem P, solutions are conventionally defined to be sequential plans rather than policies. But if a plan p is an *irredundant* solution for P, i.e., if no proper subsequence of p is a solution for P, then it is quite easy to translate p into a policy π_{p,s_0} that is equivalent to p at s_0, in the sense that both p and π_{p,s_0} produce exactly the same sequence of state transitions. We'll omit the details due to lack of space. The following lemma follows immediately:

Lemma 1 *An irredundant plan p is a solution for a classical planning problem $P = (D, \{s_0\}, G)$ iff π_{p,s_0} is a solution for P.*

Relaxations of Nondeterministic Domains. Let $D = (\mathcal{S}, \mathcal{A}, \gamma)$ be a nondeterministic planning domain whose actions are $\mathcal{A} = \{a_1, \ldots, a_n\}$. A *classical relaxation* of an action $a_i \in \mathcal{A}$ is a set of classical actions $\{a_{i1}, \ldots, a_{ik}\}$ and a state-transition function $\overline{\gamma}$ such that for every state s, $\gamma(s, a_i) = \overline{\gamma}(s, a_{i1}) \cup \ldots \cup \overline{\gamma}(s, a_{ik})$.

Let $\overline{D} = (\overline{\mathcal{S}}, \overline{\mathcal{A}}, \overline{\gamma})$ be a classical planning domain such that $\overline{\mathcal{S}} = \mathcal{S}$, and suppose there is a partition $\{\overline{A}_1, \ldots, \overline{A}_n\}$ of $\overline{\mathcal{A}}$ such that for $i = 1, \ldots, n$, $\overline{A}_i$ is a classical relaxation of the action $a_i \in \mathcal{A}$. Then $\overline{D}$ is a *classical relaxation* of D. By extension, we will say that the classical planning problem $\overline{P} = (\overline{D}, \{s_0\}, G)$ is a classical relaxation of the planning problem $P = (D, \{s_0\}, G)$.

For example, suppose D is a nondeterministic version of the Blocks World in which the action $unstack(x, y)$ has two possible outcomes: either we're holding x, or else x drops onto the table. Then we can produce a classical relaxation of D by replacing this action with an action $unstack_1(x)$ whose outcome is that we are holding x, and an action $unstack_2(x)$ whose outcome is that x is on the table.

The following theorem provides a foundation for the planning procedure in the next section. We omit the proof due to lack of space.

Theorem 1 *Let $D = (\mathcal{S}, \mathcal{A}, \gamma)$ be a nondeterministic planning domain, $P = (D, \{s_0\}, G)$ be a planning problem in D, and $\overline{P}$ be a classical relaxation of P. Suppose $\overline{P}$ has a solution plan $p = \langle a_1, \ldots, a_n \rangle$. For each a_i in p, let $\hat{a}_i$ be the action in $\mathcal{A}$ whose classical relaxation includes a_i; and let $\pi = \{(s_{i-1}, \hat{a}_i) \mid (s_{i-1}, a_i) \in \pi_{p,s_0}\}$. Then π is a weak solution for P.*

If p and π are as above, we'll call π the *policy image* of p in $\mathcal{D}$.

Procedure NDP(D, S_0, G, R)

1. $\overline{D} \leftarrow$ a classical relaxation of D; $\pi \leftarrow \emptyset$; $S_0 \leftarrow S_0 \setminus G$
2. if $S_0 = \emptyset$ then **return** π
3. **loop**
4. if $\exists s \in S_0$ s.t. $A_{\overline{D}}(s) = \emptyset$ then **return** FAILURE
5. $S \leftarrow \{$all non-goal leaf states in $\Sigma_\pi(S_0)\}$
6. if $S = \emptyset$ then
7. $\pi \leftarrow \pi \setminus \{(s, a) \in \pi | s$ isn't a π-descendant of $S_0\}$
8. **return** π
9. arbitrarily select a state $s \in S$
10. call R on the planning problem $(\overline{D}, s, G)$
11. if R returns a solution plan p then
12. $\hat{\pi} \leftarrow p$'s policy image in D
13. $\pi \leftarrow \pi \cup \{(s, a) \in \hat{\pi} \mid s \notin S_\pi\}$
14. else, R returns FAILURE
15. for every s' and a such that $s \in \gamma(s', a)$
16. $\pi \leftarrow \pi - \{(s', a)\}$
17. modify $\overline{D}$ to make the actions in a's relaxation inapplicable at s'

Figure 1: The NDP algorithm. S_π is the set of states in the policy π and $A_{\overline{D}}(s)$ is set of applicable actions in the state s given the domain $\overline{D}$.

Strong-Cyclic Planning in Nondeterministic Domains Using Classical Planners

The NDP procedure is shown in Figure 1. Its input includes a nondeterministic planning problem $P = (D, S_0, G)$, and a classical planning algorithm R. NDP successively invokes R on one or more classical planning problems, and constructs its strong-cyclic solution from R's solutions.

In Line 1, NDP generates a classical relaxation $\overline{D}$ of D, and initializes π to be the empty policy. At this point, NDP also removes any goal state in S_0 since the solution does not need to specify those states. If all initial states are goal states, NDP simply returns the empty policy (Line 2).

In Line 4, NDP checks whether there is an initial state in which there are no actions applicable; if so, there are no strong-cyclic solutions to the input planning problem.

In Line 5, NDP checks whether $\Sigma_\pi(S_0)$ contains any non-goal leaf nodes. If the answer is no, then π is a solution for P; so in line 9, NDP constructs a smaller solution for P by removing the "unreachable" parts of π, i.e, removing state-action pairs that can't be reached by executing π starting at S_0 (Line 7). Theorem 2 shows that the policy returned by NDP is a solution for P.

If $\Sigma_\pi(S_0)$ contains one or more non-goal leaf nodes, NDP arbitrarily selects one such node s and constructs a classical planning problem $(\overline{D}, \{s\}, G)$ on which to call R. When R

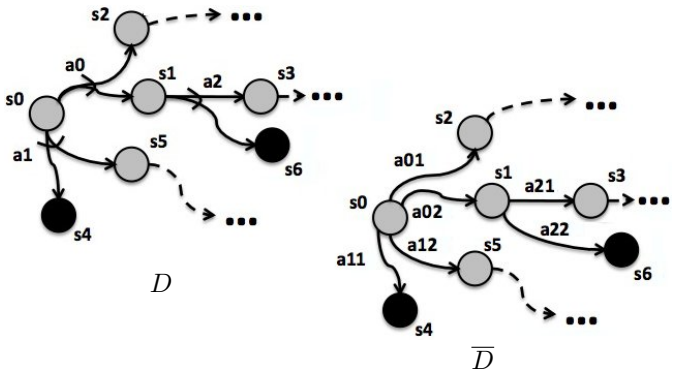

Figure 2: A nondeterministic planning domain D and its classical relaxation $\overline{D}$. Circles are states, arrows are actions, and black circles are states that have no applicable actions.

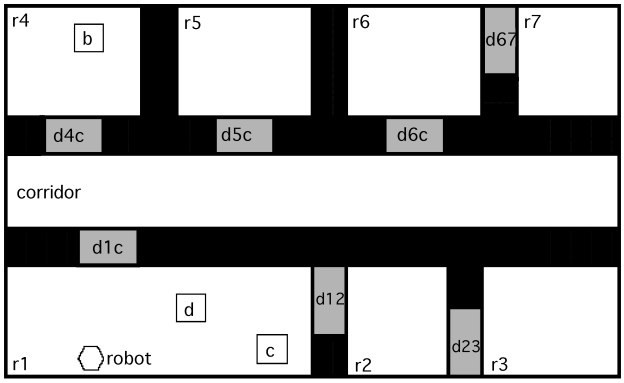

Figure 3: A state in the Robot Navigation domain.

returns, there are two cases:

- **Case 1.** R returns a solution plan p for $(\overline{D}, \{s\}, G)$. Then p's policy image $\hat{\pi}$ is a weak solution for $(D, \{s\}, G)$, so NDP incorporates into π every state-action pair $(s', a) \in \hat{\pi}$ such that s' does not already appear in π (Lines 12–13).
- **Case 2.** R returns FAILURE since it is unable to find a solution at s, which can happen if no solution exists or if R is incomplete. Hence if $s \in S_0$, then NDP returns FAILURE. Otherwise, NDP removes from π every state-action pair (s', a) such that s' is a π-parent of s, and modifies $\overline{D}$ to prevent the classical relaxation of a from being used at s' (Lines 16 –17).

Note that when NDP removes the action a specified by the current policy π for the parent s of a failed state, the parent becomes a leaf state in Σ_π in the next iteration. Since NDP modifies the domain $\overline{D}$ to make a inapplicable in s, the classical planner R does not generate a plan (if any) that starts with a in the next iteration. This provides NDP, in effect, an implicit backtracking mechanism.

As an example, consider the nondeterministic planning domain, D and its classical relaxation, $\overline{D}$, both shown in Figure 2. The circles illustrate the states in the domain, the arrows correspond to the state transitions induced by the actions, the states shown as black do not have any applicable actions, and s_0 is the only initial state.

- In NDP's first iteration, the classical planner returns the plan $\langle a_{02}, a_{21}, \ldots \rangle$, so NDP incorporates this plan in the policy π (Lines 12–13).
- s_6 is now a leaf node, so suppose NDP calls R on s_6 in the next iteration. Since no actions are applicable to s_6, R returns FAILURE. Since a_2 was the action that led to s_6, NDP modifies $\overline{D}$ to make a_2's two classical relaxations, a_{12} and a_{22}, inapplicable to s_1.
- The above change makes s_1 into leaf node; so in the next iteration, suppose NDP calls R on s_1. Since s_1 no longer has any applicable actions, R returns FAILURE, and NDP removes from $\overline{D}$ the classical relaxations a_{01} and a_{02} of the action a_0 that leads to s_1.
- There is now just one leaf node, namely s_0. Hence in the next iteration, NDP calls R on s_0. This time, R returns a plan that includes a_{12}. Thus NDP incorporates a_1 into its policy, which makes s_4 a new leaf node.
- Since s_4 has no applicable actions, what eventually will happen is that R will eventually return FAILURE at s_4, NDP will remove a_{11} and a_{12} so that s_0 now has no applicable actions. Hence R will return FAILURE at s_0, and at this point NDP will return FAILURE too.

The following theorem (the proof is omitted due to lack of space) establishes conditional soundness and completeness for NDP. In this theorem (and throughout the rest of the paper), the notation NDP$/R$ refers to using NDP with R as its classical planning algorithm.

Theorem 2 *If R is sound and/or complete, so is* NDP$/R$.

Conjunctive Abstraction

In some cases, a nondeterministic action may have a huge number of possible outcomes. Thus, NDP may make a huge number of calls to its classical planner in order to generate a solution policy, and in the worst case, the number of such calls will be exponential in the size of the domains.

For example, consider the Robot Navigation Domain, which is often used as a benchmark problem for nondeterministic planning domains (Cimatti et al. 2003). This domain is a variant of a similar domain described in (Kabanza, Barbeau, and St-Denis 1997). It consists of a building with 8 rooms connected by 7 doors (see the map in Figure 3), where several objects that need to be moved to desired locations, and a robot that is responsible for moving them. The robot can open and close doors, move through doorways, and pick up and put down objects. It can hold at most one object at a time. Some or all of the doors are "kid doors," and a "kid" can open or close any combination of those doors after each of the robot's actions. The kid is modeled not as a separate agent, but as a set of nondeterministic effects for each action.

In the Robot Navigation domain with k "kid doors," opening a door has 2^k possible outcomes (2^k possible combinations of open and closed kid doors). Thus it would take $\Theta(2^k)$ space to write the action as a nondeterministic

STRIPS operator, it would require to make $\Theta(2^k)$ calls to the classical planner R from NDP, and therefore, solving planning problems that contain this operator would take time and space $\Omega(2^k)$. The MBP planner (Cimatti et al. 2003) tackles this problem by using Binary Decision Diagrams (BDDs) to encode sets of states and planning operators that operate over those sets.

In NDP, we cannot exploit a BDD-based (or a similar) machinery, because NDP does not know the internals of its classical planner R, nor can it modify R to use BDDs. Instead, we developed a way to accomplish a similar (though less powerful) effect within a conventional nondeterministic-STRIPS-operator representation, by changing some of the planning operators' effects and introducing one new planning operator. These modifications provide a way to go back and forth, in a controlled fashion, between individual states and classes of states during planning.

We start with a planning domain D using the conventional state representation in which a state s is a set of ground atoms, and a ground atom is true iff it is in s. Let Q be the set of all ground atoms in D. Let S be the set of all states in which one set of atoms $Q_T \subset Q$ are each true, another set of atoms $Q_F \subset Q$ are each false. Let $Q_I = Q - Q_T - Q_F$. Then S contains $2^{|Q_I|}$ states such that for each atom $q \in Q_I$, there is a state $s \in S$ in which q is true and another state $s' \in S$ in which q is false.

We'll now define $\hat{D}$, D's *conjunctive abstraction domain* (so called because conjuncts represent sets of states). $\hat{D}$'s initial states and goal states are the same as in D. $\hat{D}$'s atoms and operators are as follows:

The atoms. In $\hat{D}$, the set of ground atoms $\hat{Q}$ consists of Q, plus a new ground atom ignore-q for each atom $a \in Q$. Here, ignore-q's intended meaning is that there are two states s, s' in S such that q is true in s and it is false in s'. In D, we can represent S as a single state, namely $\hat{S} = Q_T \cup \{\text{ignore-}q \mid q \in Q_I\}$.

As an example, consider Fig. 3 again. If we don't care which doors are open and which doors are closed, then the figure corresponds to a set S_{Fig2} of 2^7 different states. In the conjunctive abstraction domain, the abstract state $\hat{S}_{\text{Fig2}}$ is

{loc(robot, r1), loc(c, r1), loc(d,r1), loc(b,r4), ignore-open(d1c), ignore-open(d12), ignore-open(d23), ignore-open(d4c), ignore-open(d5c), ignore-open(d6c), ignore-open(d67) } .

It is important to note that, S does not represent uncertainty in the state knowledge and thus, its semantics are quite different a "belief state " as in partially observable planning. Instead, S represents sets of those states can be clustered together because searching over them together is equivalent to searching over each of them individually in terms of generating a solution to a fully observable nondeterministic planning problem.

The operators. We can translate a planning operator o of D into an operator $\hat{o}$ that produces an abstract state, by adding an effect ignore-q if o's effects don't already include q or $\neg q$. Whether we'll prefer to do this or to retain o unmodified will depend on the planning domain.

Let S be a set of states and $\hat{S}$ be S's abstract state. If an operator o has a precondition q, we cannot apply o in $\hat{S}$ unless we know whether q is true or false. For example, in the Robot Navigation Domain, we can't move through a doorway unless we know whether the doorway is open or closed. To get q's truth value, we may need to split S into subsets. To enable this to happen, $\hat{D}$ will include, for each predicate p in D, a *splitting operator* split-p.

split-$p(x_1, \ldots, x_n)$
precond: ignore-$p(x_1, \ldots, x_n)$,
effects1: $\neg$ignore-$p(x_1, \ldots, x_n), p(x_1, \ldots, x_n)$
effects2: $\neg$ignore-$p(x_1, \ldots, x_n), \neg p(x_1, \ldots, x_n)$

For example, split-open(d1c) splits $\hat{S}_{\text{Fig2}}$ into two abstract states that represent $\{s \in S \mid \text{open(d1c) is true}\}$ and $\{s \in S \mid \text{open(d1c) is false}\}$. This tells NDP to reason separately about what to do when the door is open and when it is closed.

Note that although splitting operators resemble nondeterministic planning operators syntactically, their semantics is quite different. First, since their intent is to manage abstract states, they *do not appear* in the solution policies returned by NDP. Second, their possible outcomes *do not* model nondeterminism; instead they're used to tell NDP to plan for all possible truth values of some atom.

Experimental Evaluation

We implemented NDP in Common Lisp, and compared it with MBP on three fully-observable nondeterministic planning domains that are well-known from previous experimental studies: Robot Navigation (Cimatti et al. 2003), Hunter-Prey (Kuter et al. 2005)[2] , and a nondeterministic version of the Blocks World (Kuter and Nau 2004). All of these domains admit strong-cyclic solutions.

For NDP, we used the conjunctive abstraction scheme described in the previous section; but we did not use this scheme for MBP. MBP already uses BDDs, a more powerful abstraction scheme, and our preliminary trials showed that MBP performed much worse with conjunctive abstraction than without it.[3]

For NDP's classical planner, we used the most recent available versions of the four well-known classical planners: FF (Hoffmann and Nebel 2001), LPG (Gerevini, Saetti, and Serina 2003), SatPlan (Kautz, Selman, and Hoffmann 2006), and SGPlan (Hsu et al. 2006).

Recall that "NDP/SGPlan" means NDP using SGPlan, "NDP/FF" means NDP using FF, and so forth. We ran the experiments on a Macbook Pro with a 2.16GHz Intel Core Duo processor, running Fedora Core 6 Linux on a virtual machine with 512MB memory.[4]

[2]The Hunter-Prey domain is a "moving target" problem first introduced in (Koenig and Simmons 1995) as a testbed for real time search, in which planning and execution may sometimes be interleaved. As demonstrated in (Kuter et al. 2005), it is also a non-trivial benchmark for offline generation of solutions for nondeterministic planning problems; thus, we also used it here.

[3]With conjunctive abstraction, MBP had two ways to create abstract states (its BDDs and our operators), and its search algorithm

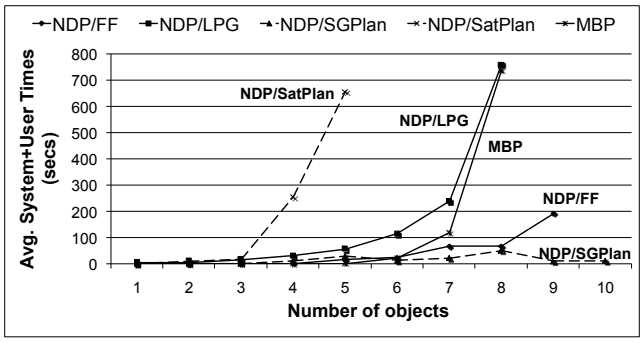

Figure 4: Average running times in seconds in Robot-Navigation with 7 kid doors, as a function of the number of objects.

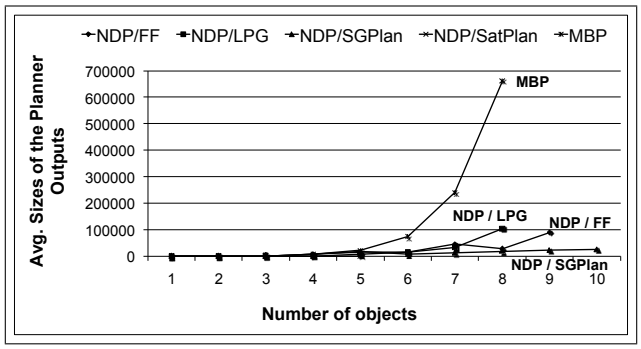

Figure 5: Average solution size in Robot-Navigation with 7 kid doors, as a function of the number of objects. The solution sizes of NDP/FF, NDP/LPG, NDP/SatPlan, and NDP/SGPlan were nearly equal, hence the label NDP/R refers to all of them.

Robot Navigation. The first set of experiments were in the Robot Navigation domain described previously, with $k = 7$ (i.e., all 7 doors were kid doors). We varied the number of objects n from 1 to 10. For each value of n, we measured each planner's average CPU time on 20 randomly-generated problems.[5] As shown in Figure 4, the best running times were achieved by NDP/FF and NDP/SGPlan, followed by NDP/LPG and MBP, followed by NDP/SatPlan.

MBP cannot use the search heuristics in the classical planners, hence it searched most of the state space in most of the planning problems. The heuristics in NDP/SGPlan and NDP/FF avoided visiting most of the state space, hence they had much better running times than MBP.

We were surprised that NDP/SatPlan performed so badly. We inspected the CNF formulas that SatPlan produces while planning, and found that their sizes were on the order of 10MB in this domain. As SatPlan always attempts to generate optimal plans, this leads us to suspect that SatPlan was generating a CNF representation of most of the state space at the beginning of the planning process and/or making excessive use of splitting operators to generate lots of non-abstract states during that process.

Figure 5 shows the average sizes of the solutions generated by the planners in these experiments, where size is measured as the number of terms in the policy representations produced by the planners. In our opinion, size is a very good measure of solution quality—but to the best of our knowledge, no good measure of solution quality exists for nondeterministic planning problems. One might like to measure something like a solution's average-case or worst-case execution length, but the average-case execution length is undefined since there are no probabilities on the actions' outcomes, and if the solution contains a cycle, the worst-case execution length has no upper bound.[6]

As shown in Figure 5, MBP's solutions were much larger than the ones found by NDP. One reason for this is that the boolean formulas in MBP's solutions weren't in simplest form: MBP has an option for doing boolean simplification on its BDD representations, but we didn't use this option because it increases MBP's running time. Another reason is that the solution policies produced by MBP tell what actions to perform even in states that can never be reached in any execution of the policy. If we could somehow remove the parts of the BDD that represent these "unreachable" states and then do boolean simplification, then we suspect the resulting solution size might be similar to NDP's.

Hunter-Prey. In our version of the Hunter-Prey domain, the world is an $n \times n$ grid in which a hunter wants to catch one or more prey. The world is fully observable: the hunter can always observe the locations of the prey. The hunter has five possible actions; move north, south, east, or west, and catch (the latter is applicable only when the hunter and prey are in the same location). The prey has also five actions: the four movement actions plus a stay-still action. But (analogously to the kid doors in the Robot Navigation domain), the prey is not represented as a separate agent: instead, its possible actions are encoded as nondeterministic outcomes for the hunter's actions.

Figure 6 shows the running time for each planner when there is just one prey and the grid size varies from 2×2 to 6×6. Each data point is the average of 20 randomly generated problems. In some cases, NDP's classical planner had a memory overflow or could not solve a problem within our time limit (20 minutes), and in those cases we omitted the corresponding data point.

MBP's running time were significantly better than NDP's, regardless of which classical planner NDP used.

insisted on visiting *both* sets of states.

[4] In all these experiments, we ran LPG with its "-speed" option. We also did not use LPG's default value for the maximum variables that could a planning problem have in order to avoid memory overflow problems even in small problem instances in our experiments.

[5] As in (Pistore and Traverso 2001), MBP's CPU times include both its preprocessing and search times. Omitting the former would not have significantly affected the results: they were never more than a few seconds, and usually below one second.

[6] In other words, for every finite number n, there is an execution of length greater than n. It would be tempting to conclude from this that the worst-case execution time is infinite, but this wouldn't be quite correct since every fair execution has finite length.

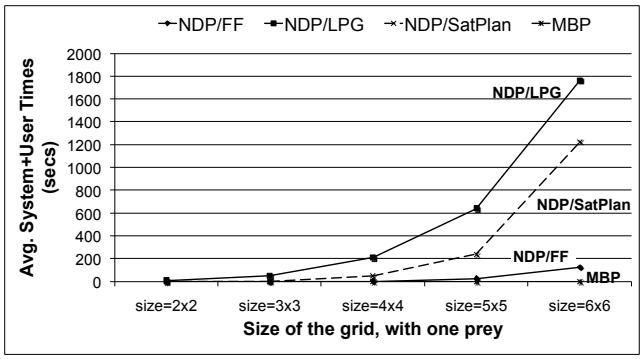

Figure 6: Average running times in seconds in Hunter-Prey with one prey, as a function of grid size.

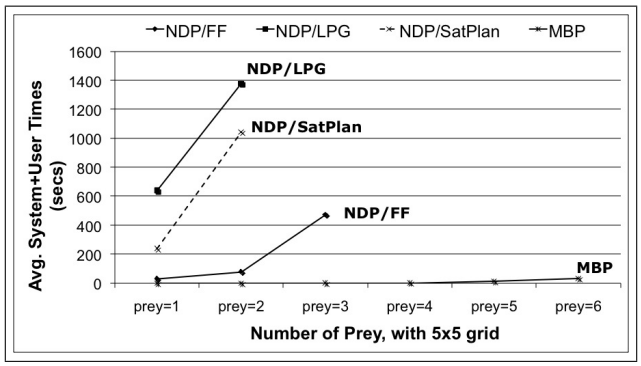

Figure 7: Average running times in seconds in Hunter-Prey as a function of the number of prey, on a 5×5 grid.

This occurred because in the Hunter-Prey domain, MBP's BDD-based abstract states did a much better job of compressing the search space than conjunctive abstraction did. For example, MBP could use a single BDD to represent the set of all states in which the hunter needed to move in a particular direction, but NDP could not represent these states as a single abstract state because the states did not have enough ignore atoms in common.

To investigate this further, we have also done experiments with varying number of prey in fixed grid. As shown in Figure 7, MBP still did best when we fixed the grid size at 5×5 and varied the number of prey from 2 to 6. On the larger problems, NDP/FF and NDP/LPG exceeded our 20-minute time limit, and NDP/SatPlan and NDP/SGPlan generated error messages. We think these messages occurred during SatPlan's and SGPlan's preprocessing phases, but we did not investigate this in detail since it would have required modifying the planners' source code.

Nondeterministic Blocks World. The nondeterministic Blocks World is like the classical Blocks World, except that an action may have three possible outcomes: (1) the same outcome as in the classical case, (2) the block slips out of the gripper and drops on the table, and (3) the action fails completely and the state does not change.

Figure 8 shows the planners' average CPU times in this

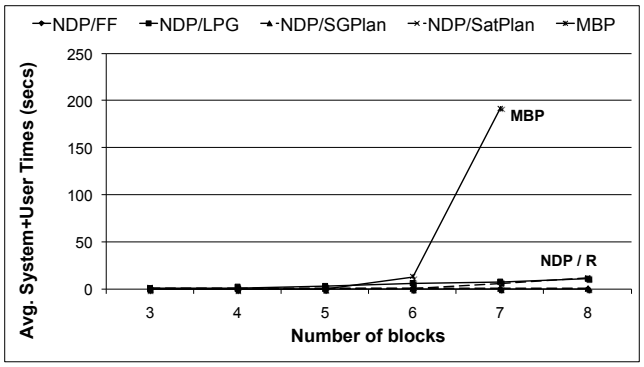

Figure 8: Average running times in seconds in the nondeterministic Blocks World, as a function of the number of blocks. The running times of NDP/FF, NDP/LPG, NDP/SatPlan, and NDP/SGPlan were nearly equal, hence the label NDP/R refers to all of them.

domain, as a function of the number of blocks. Each data point represents the average running time on 20 random problems. MBP did badly compared to NDP regardless of which classical planner NDP used. The reasons are twofold. First, there were no large sets of states that could be clustered together; hence neither MBP's BDD-based representation nor conjunctive abstraction could make much difference. Second, MBP did not exploit the heuristics used in the classical planners, hence MBP searched the most of the state space in most planning problems.

Related Work

Early works on fully-observable nondeterministic domains include the CASSANDRA planning system (Pryor and Collins 1996), CNLP (Peot and Smith 1992), PLINTH (Goldman and Boddy 1994), and UCPOP (Penberthy and Weld 1992), and QBFPLAN (Rintanen 1999). These algorithms do not perform as well as planners such as MBP, and cannot scale up to large planning problems.

One of the earliest attempts to use model-checking techniques for planning under nondeterminism is first introduced in the SIMPLAN planner of (Kabanza, Barbeau, and St-Denis 1997). SIMPLAN is based on model checking techniques that work over explicit representations of states in the state space; i.e., the planner represents and reasons explicitly about every state visited during the search. Symbolic model-checking techniques, such as Binary Decision Diagrams (BDDs), to do planning in nondeterministic domains under the assumptions of fully-observability and classical reachability goals were first introduced in (Giunchiglia and Traverso 1999; Cimatti et al. 2003). MBP is one of the best planners that uses BDDs for this purpose.

UMOP (Jensen, Veloso, and Bowling 2001; Jensen, Veloso, and Bryant 2003) combines BDDs with a heuristic-search algorithm for strong and strong-cyclic planning (Jensen, Veloso, and Bryant 2003). Heuristic search provides some performance improvements over unguided BDD-based planning on some toy examples (Jensen, Veloso,

and Bryant 2003); the authors discuss how the approach would scale up to real-world planning.

Conformant planning is a special case of nondeterministic planning in which the goal is to generate a linear sequence of actions (rather a than policy) with no observations. There are many nondeterministic planning problems that have strong and/or strong-cyclic solutions but don't have conformant solutions. The conformant-planning approach closest to our work is (Palacios and Geffner 2006; 2007), which describes how to translate any conformant planning problem into a classical problem for a classical planner. Their approach generates a single translation (rather than a sequence of translations as we do), and can only find conformant solutions.

Planners such as MBP (Bertoli et al. 2006), POND (Bryce, Kambhampati, and Smith 2006) and Contingent-FF (Hoffmann and Brafman 2005) are developed for partial observable planning, where actions may or may not have nondeterministic outcomes and the state of the world is only partially observable during the execution of a plan. These planners cannot generate cyclic solutions; instead, they generate acyclic solution graphs. (Bertoli, Cimatti, and Pistore 2006) demonstrated cases of partial observability in which it is possible to generate strong-cyclic solutions, but the definition of such solutions does not apply in the conventional fully observable case.

FF-Replan (Yoon, Fern, and Givan 2007) uses the FF planner (Hoffmann and Nebel 2001) to first generate a plan (i.e., a weak policy) for a *determinized* version of a Markov Decision Process (MDP), similar to our classical relaxation schema in this paper. However, FF-Replan does not produce strong-cyclic policies that guarantee to reach a goal state despite nondeterminism, as we do. If the simulated execution of a weak plan reaches a goal state, the system does not perform any further planning (even though other executions would not necessarily reach a goal state). Otherwise, if the execution produces an unexpected state, then FF-Replan attempts to generate another weak policy from that state to the goal state, until a successful execution occurs.

Other approaches to *reactive planning*, including (Schoppers 1987; Haigh and Veloso 1996; Despouys and Ingrand 1999) are used in situations where it is unrealistic to compute in advance how to react to all of the possible outcomes of an action. These approaches generally interleaves planning and execution; NDP, on the other hand, is designed for generating offline policies. The early work on "universal planning" (Schoppers 1987) depends on backward-chaining from the given goals towards an initial state (or to an observed current situation), in order to perform goal regression similar to early STRIPS. An important point difference between this work and NDP is that the latter is a forward-chaining state-space planner, which enables it to know the state of the world at all times. This allows using any classical planner as R in our abstract planning procedure.

Finally, (Fox et al. 2001) reports an approach for analyzing deterministic planning domains and identifying structural features and dependencies among those features using model-checking techniques. Although this approach has some similarities to our conjunctive abstraction technique, their approach focusing of using the results of a domain analysis to prune search space whereas we use conjunctive abstraction for state-space compression. It would be interesting to investigate as a future work if the domain analysis method can be used for identifying more general and effective features for state compression.

Conclusions

We have described NDP, an algorithm that can use any classical planner R to generate strong-cyclic solutions to nondeterministic planning problems by calling R on a sequence of classical planning problems and combining R's results on those problems. We have presented theoretical results on NDP's soundness and completeness.

We also have described *conjunctive abstraction*, a way to alleviate exponential blowup in nondeterministic planning by rewriting the planning operators to operate over sets of states represented as conjuncts of literals. While not as powerful as MBP's BDDs, conjunctive abstraction has the advantage that it can be used with any planner that uses a nondeterministic-STRIPS-operator representation, without modifying the planner.

In our experiments with four different classical planners, FF was the one that worked best with NDP and conjunctive abstraction. SGPlan also did quite well (except in the Hunter-Prey domain, where it generated error messages in most cases). The classical planner that did worst with NDP was SatPlan; and from our examination of SatPlan's output, we suspect it was trying to reason about individual states despite the availability of the abstract ones.

Discussion. NDP's advantage over MBP is that MBP uses none of the sophisticated search heuristics used in classical planners. In the Robot Navigation domain and the Nondeterministic Blocks World, the search heuristics of FF and SGPlan worked well, hence NDP/FF and NDP/SGPlan did much better than MBP.

On the other hand, conjunctive abstraction can't provide much state-space compression as BDDs if we want to plan the same actions in every state that satisfies some disjunctive condition $c_1 \vee c_2 \vee \ldots \vee c_n$. In the Hunter-Prey domain, this enabled MBP to outperform NDP regardless of which classical planner we used. In general, the planning domains where conjunctive abstraction works well are the ones where for each state s and action a, the set of states S' in which a is the "next action" corresponds either to a single conjunct of literals or a small number of conjuncts of literals. Our experimental domains were chosen to illustrate this point.

Future work. The above considerations suggest a promising direction for future work: even better performance may be achievable by writing an NDP-like planner that incorporates an FF-like algorithm operating over BDDs. Such a planner should be able to outperform both MBP and NDP in all four of the experimental domains.

In such a planner, even further improvements could be achieved by more tightly coupling the NDP and FF algorithms. For example, when the current NDP algorithm calls

FF, it must wait until FF reaches a goal. If we could intervene to stop FF as soon as it reaches a state that is already part of NDP's current partial solution, this would provide a substantial speedup because it would prevent FF from wasting time retracing large parts of the solutions that it found during the previous times NDP called it.

Thirdly, we note that some MDP planning algorithms such as LAO* (Hansen and Zilberstein 2001) can generate cyclic solution policies. With proper modifications to these planners and their inputs, it will be interesting to compare them with NDP and classical planners. This may provide a path toward developing an NDP-like algorithm for MDPs.

Finally, we want to investigate the relationship of NDP to planners such as Contingent-FF and POND that generate acyclic solutions to partially observable planning problems. We intend in the near future to develop a modified version of NDP that produces acyclic solutions and to compare it experimentally to those planners.

Acknowledgments. This work was supported in part by DARPA's Transfer Learning and Integrated Learning programs, NSF grant IIS0412812, and AFOSR grants FA95500510298, FA95500610405, and FA95500610295. The opinions in this paper are those of the authors and do not necessarily reflect the opinions of the funders.

References

Bertoli, P.; Cimatti, A.; Roveri, M.; and Traverso, P. 2006. Strong Planning under Partial Observability. *Artificial Intelligence* 170:337–384.

Bertoli, P.; Cimatti, A.; and Pistore, M. 2006. Strong Cyclic Planning under Partial Observability. *ECAI*.

Bryce, D.; Kambhampati, S.; and Smith, D. E. 2006. Planning Graph Heuristics for Belief Space Search. *Journal of Artificial Intelligence Research* 26:35–99.

Cimatti, A.; Pistore, M.; Roveri, M.; and Traverso, P. 2003. Weak, strong, and strong cyclic planning via symbolic model checking. *Artificial Intelligence* 147(1-2):35–84.

Despouys, O., and Ingrand, F. 1999. Propice-Plan: Toward a unified framework for planning and execution. In *ECP*.

Fox, M.; Long, D.; Bradley, S.; and McKinna, J. 2001. Using model checking for pre-planning analysis. In *Proceedings of the AAAI Symposium on Model-based Validation of Intelligence*.

Gerevini, A.; Saetti, A.; and Serina, I. 2003. Planning through Stochastic Local Search and Temporal Action Graphs. *JAIR* 20:239–290.

Giunchiglia, F., and Traverso, P. 1999. Planning as model checking. In *ECP*.

Goldman, R. P., and Boddy, M. S. 1994. Conditional linear planning. In *AIPS*.

Haigh, K. Z., and Veloso, M. 1996. Interleaving planning and robot execution for asynchronous user requests. In *Planning with Incomplete Information for Robot Problems: Papers from the 1996 AAAI Spring Symposium*, 35–44. AAAI Press.

Hansen, E., and Zilberstein, S. 2001. LAO*: A Heuristic Search Algorithm that Finds Solutions with Loops. *Artificial Intelligence* 129:35–62.

Hoffmann, J., and Brafman, R. 2005. Contingent Planning via Heuristic Forward Search with Implicit Belief States. In *ICAPS*.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14:253–302.

Hsu, C. W.; Wah, B. W.; Huang, R.; and Chen, Y. X. 2006. New Features in SGPlan for Handling Soft Constraints and Goal Preferences in PDDL3.0.

Jensen, R.; Veloso, M. M.; and Bowling, M. H. 2001. OBDD-based optimistic and strong cyclic adversarial planning. In *ECP*.

Jensen, R.; Veloso, M. M.; and Bryant, R. 2003. Guided symbolic universal planning. In *ICAPS*.

Kabanza, F.; Barbeau, M.; and St-Denis, R. 1997. Planning control rules for reactive agents. *Artificial Intelligence* 95(1):67–113.

Kautz, H.; Selman, B.; and Hoffmann, J. 2006. SatPlan: Planning as Satisfiability.

Koenig, S., and Simmons, R. G. 1995. Real-time search in non-deterministic domains. In *IJCAI-1995*.

Kuter, U., and Nau, D. 2004. Forward-chaining planning in nondeterministic domains. In *AAAI-2004*.

Kuter, U.; Nau, D.; Pistore, M.; and Traverso, P. 2005. A hierarchical task-network planner based on symbolic model checking. In *ICAPS*.

Palacios, H., and Geffner, H. 2006. Compiling Uncertainty Away: Solving Conformant Planning Problems Using a Classical Planner (Sometimes). In *AAAI*.

Palacios, H., and Geffner, H. 2007. From Conformant into Classical Planning: Efficient Translations that may be Complete Too. In *ICAPS*.

Penberthy, J. S., and Weld, D. 1992. UCPOP: A Sound, Complete, Partial Order Planner for ADL. In *KR*.

Peot, M., and Smith, D. 1992. Conditional nonlinear planning. In *AIPS*.

Pistore, M., and Traverso, P. 2001. Planning as Model Checking for Extended Goals in Non-deterministic Domains. In *IJCAI*.

Pryor, L., and Collins, G. 1996. Planning for contingency: a decision based approach. *JAIR* 4:81–120.

Rintanen, J. 1999. Improvements to the evaluation of quantified boolean formulae. In *IJCAI*.

Schoppers, M. 1987. Universal plans for reactive robots in unpredictable environments. In *IJCAI*, 1039–1046.

Yoon, S.; Fern, A.; and Givan, R. 2007. FF-Replan: A Baseline for Probabilistic Planning. In *ICAPS*.