Forward-Chaining Planning in Nondeterministic Domains

Ugur Kuter
Department of Computer Science,
University of Maryland,
College Park, MD 20742, USA
email: ukuter@cs.umd.edu

Dana Nau
Department of Computer Science and
Institute for Systems Research
University of Maryland,
College Park, MD 20742, USA
email: nau@cs.umd.edu

Abstract

In this paper, we present a general technique for taking forward-chaining planners for deterministic domains (e.g., HSP, TLPlan, TALplanner, and SHOP2) and adapting them to work in nondeterministic domains. Our results suggest that our technique preserves many of the desirable properties of these planners, such as the ability to use heuristic techniques to achieve highly efficient planning.

In our experimental studies on two problem domains, the well-known MBP algorithm took exponential time, confirming prior results by others. A nondeterminized version of SHOP2 took only polynomial time. The polynomial-time figures are confirmed by a complexity analysis, and a similar complexity analysis shows that a nondeterminized version of TLPlan would perform similarly.

Introduction

One of the biggest limitations of classical AI planning is the assumption of *determinism*: classical planner assumes that it knows the exact outcomes of the actions, so that for any given plan and initial state, the world will evolve along a single fully predictable path.

A more realistic assumption is that the world is *nondeterministic*: an action may have several possible outcomes, but we do not know in advance which one will occur. Unfortunately, this incurs a huge combinatorial explosion: a plan may have exponentially many different execution paths, and the planning algorithm must reason about all of them in order to find a plan that works despite the nondeterminism in the world.

Prior approaches for planning in nondeterministic domains have included model-checking techniques (Cimatti *et al.* 2003; Cimatti, Roveri, & Traverso 1998) and conformant-planning techniques (Smith & Weld 1998). Algorithms based on these approaches typically examine most or all of the states in the state space.

In deterministic planning domains, much work has been done on ways to improve the efficiency of planners by preventing them from visiting unpromising states. This work has been especially successful in planners based on forward chaining, such as HSP (Bonet & Geffner 1999), FF (Hoffmann & Nebel 2001), TLPlan (Bacchus & Kabanza 2000), TALplanner (Kvarnström & Doherty 2001), and SHOP2 (Nau *et al.* 2003). These planners know the current state at all times, which facilitates the use of some powerful pruning techniques—especially in hand-tailorable planners, in which the pruning techniques may be domain-specific.

In this paper, we present a way to transport these efficiency improvements over to nondeterministic planning. Our contributions are the following:

- A general technique for "nondeterminizing" forward-chaining planners, i.e., extending them to work in nondeterministic domains. This technique preserves soundness and completeness.
- Conditions under which a nondeterminized algorithm's time complexity is polynomially bounded by the time complexity of the deterministic version.
- Experimental comparisons of ND-SHOP2 (the nondeterminization of SHOP2) to the well known MBP algorithm, in two different domains. The experimental results show MBP's CPU time growing exponentially in the size of the problem, confirming the results in (Pistore, Bettin, & Traverso 2001), and ND-SHOP2's CPU time growing only polynomially.
- A complexity analysis confirming the experimental results for ND-SHOP2: its running time grows at $\Theta(n^4)$ in one domain and $\Theta(n^5)$ in the other. This level of performance is not restricted to ND-SHOP2: a similar complexity analysis shows that a nondeterminization of TLPlan would have similar growth rates.

Definitions and Notation

Our definition of states, goals, operators, and actions (ground instances of operators) are the same as in classical planning, except that each operator (and thus each action) may have more than one possible result. A planning problem is a triple $P = (s_0, g, O)$, where s_0 is the initial state, g is the goal, and O is the set of operators. Recall that s_0 and g vary from one problem to another, and O remains fixed within a planning domain

but varies from one domain to another.

We also use most of the same definitions used in nondeterministic planning domains. $\gamma(s,a)$, the *state-transition function* induced by the set of operators O, gives the set of all possible states that may result from applying the action a to the state s, with $\gamma(s,a) = \emptyset$ if a is not applicable to s. The planning domain is deterministic iff $|\gamma(s,a)| \leq 1\ \forall s \forall a$. A *policy* is a set $\pi = \{(s_i, a_i)\}_{i=1}^{k}$, where $k \geq 0$ is the size of the policy. The set of states in π is $S_\pi = \{s \mid (s,a) \in \pi\}$. We require that for any state s, there is at most one action a such that $(s,a) \in \pi$. The *execution structure* for π is a directed graph Σ_π whose nodes are all of the states that can be reached by executing actions in π, and whose edges represent the possible state transitions caused by actions in π. If there is a path in Σ_π from s_1 to s_2, then we say that s_1 is a π-ancestor of s_2 and s_2 is a π-descendant of s_1. We use the usual definitions (Cimatti *et al.* 2003) of weak, strong, and strong-cyclic solutions for planning problems.

Researchers have often defined nondeterministic versions of well-known deterministic problems; we formalize this notion as follows. A planning problem $P' = (S_0, g, O')$ is a *nondeterministic version* of a deterministic planning problem $P = (s_0, g, O)$ if $s_0 \in S_0$ and O is identical to O' except that each operator in O' may have additional sets of effects. These additional effects can be used to model action failures (e.g., a gripper drops what it is holding) and exogenous events (e.g., a road is closed).

Nondeterminizing Forward Planners

We now describe a general technique for taking forward-chaining planners for deterministic planning domains, and *nondeterminizing* them, i.e., translating them into planners that find strong-cyclic solutions in nondeterministic domains. A policy π is a *strong-cyclic solution* if every (finite or infinite) path in Σ_π can be extended to a finite execution path that reaches to a goal state. We have also developed planners that produce weak and strong solutions; see (Kuter & Nau 2004) for details.

The basis of our approach is an abstract planning procedure FCP for deterministic domains, and a corresponding procedure ND-FCP for nondeterministic domains. Both are described below. If a planner Λ can be described as an instance of FCP, then, ND-Λ, the corresponding instance of ND-FCP, will be the strong-cyclic nondeterminization of Λ. Using this technique, we have written strong-cyclic nondeterminizations of TLPlan (Bacchus & Kabanza 2000), SHOP2 (Nau *et al.* 2003), TALplanner (Kvarnström & Doherty 2001), and several other forward planners (see (Kuter & Nau 2004)).

As shown in Figure 1, FCP starts with an initial state s_0, and explores other states by successively choosing and applying planning operators until it reaches a state that satisfies the goal formula g. The *action-selection function* $\alpha(s)$ is used to prune the search space. It returns zero or more actions, which we will call the actions that *satisfy* $\alpha(s)$. Rather than trying all of the actions applicable to s, FCP only tries the ones that are applicable and satisfy α.

```
Procedure FCP(s_0, g, O, α);
  π ← ∅; s ← s_0
  loop
    if s satisfies g then return(π)
    A ← {(s,a) | a is a ground instance of an operator
          in O, a is applicable to s, and a ∈ α(s)}
    if A = ∅ then return(failure)
    nondeterministically choose (s,a) ∈ A
    π ← π ∪ {(s,a)}
    s ← γ(s,a)
```

Figure 1: An abstract forward-chaining planning procedure for deterministic planning domains.

```
Procedure ND-FCP(S_0, g, O', α')
  π ← ∅; S ← S_0; solved ← ∅
  loop
    if S = ∅ then return(π)
    select a state s ∈ S and remove it from S
    if s satisfies g then insert s into solved
    else if s ∉ S_π then
      A ← {(s,a) | a is a ground instance of an opera-
            tor in O', a is applicable to s, and
            a ∈ α'(s)}
      if A = ∅ then return(failure)
      nondeterministically choose (s,a) ∈ A
      π ← π ∪ {(s,a)}
      S ← S ∪ γ(s,a)
    else if s has no π-descendants in (S ∪ solved) \ S_π
      then return(failure)
```

Figure 2: Nondeterminization of FCP. The underlines indicate how the coding from FCP is embedded in ND-FCP.

For example, in SHOP2, the search is controlled by a set of *methods* that decompose *tasks* into subtasks, and $\alpha(s)$ is the set of all actions a applicable to s such that a can be produced by applying methods to the current task network. In TLPlan (Bacchus & Kabanza 2000), the search is controlled by a formula ϕ written in a modal logic called *Linear Temporal Logic (LTL)*. For each state s, $\alpha(s)$ is the set of all actions a applicable to s such that the successor state $\gamma(s,a)$ satisfies a *progressed formula*, $Progress(s, \phi)$.

Figure 2 shows ND-FCP, the strong-cyclic nondeterminization of FCP. Like FCP, ND-FCP plans forward from S_0, but the success criterion is more complicated. In FCP, a plan π is a solution if its final state satisfies the goal condition g. In order for a policy in ND-FCP to be a solution, every state in π's execution structure must have at least one π-descendant that satisfies the goal (a *solved* state in the pseudocode).

ND-FCP's action-selection function is analogous to

FCP's, and the action-selection function for a deterministic domain can generally be adapted to work in nondeterministic versions of the domain. As an example, recall that for TLPlan, the action-selection function $\alpha(s)$ for a planning domain $\mathcal{D}$ returns all actions applicable to s such that $\gamma(s,a)$ satisfies $Progress(s,\phi)$, where ϕ is a search-control formula. If $\mathcal{D}'$ is a nondeterministic version of $\mathcal{D}$, then one possible action-selection function for ND-TLPlan is $\alpha'(s) = \{$all actions a such that at least one state in $\gamma(s,a)$ satisfies $Progress(s,\phi)\}$. This trivial adaptation of α works correctly, but usually ND-TLPlan will perform more efficiently if we write an action-selection function that returns only some of the actions in $\alpha'(s)$.

Theoretical Properties

It is not hard to show that FCP and ND-FCP are both sound (i.e., they do not return any plan that is not a solution), and that they are conditionally complete in the sense that they can find every solution whose actions satisfy the action-selection function.

We now establish an upper bound on the time complexity of a nondeterminized planning algorithm in a strongly connected planning domain. A planning domain is strongly connected if for any two states s and s', there exists a sequence of actions that reaches to s' when executed in s. Such domains are not hard to find: some well-known examples from previous planning competitions include Blocks-World, Logistics, DriverLog, ZenoTravel, Depot, and Rover. Any nondeterminization of such a domain will also be strongly connected.

Theorem 1 *Let Λ be an instance of FCP. Suppose Λ finds solution plans in time $O(\rho(|\pi|))$ in strongly-connected planning domains, where $|\pi|$ is the size of the solution plan and ρ is a monotonic function. Then* ND-Λ *finds solution policies in time $O(\rho(|\Sigma_{\pi'}|))$, where $|\Sigma_{\pi'}|$ is the size of execution structure for the solution policy returned by* ND-Λ.

Corollary 1 *Under the conditions of Theorem 1, if the number of possible successors of each state is bounded by a constant, then* ND-Λ *finds solution policies in time $O(\rho(|\pi'|))$, where $|\pi'|$ is the size of the solution policy.*

Note that every path in $\Sigma_{\pi'}$ can be extended to a finite execution path that reaches to a goal state in a strongly-connected domain. In domains that are not strongly connected, however, $\Sigma_{\pi'}$ may contain states that have no π'-descendants that satisfy the goals. We call such states as the *dead-end* states of $\Sigma_{\pi'}$.

The following complexity result holds even in planning domains that are not strongly connected:

Theorem 2 *Let Λ be an instance of FCP, and suppose that Λ's running time is $O(\rho(|\pi|))$, where $|\pi|$ is the size of the solution plan returned by Λ, and ρ is a monotonic function. Then* ND-Λ *finds solution policies in average time $O(\rho(n) + \frac{xdn}{y}))$, where $n = |\Sigma_{\pi'}|$ is the size of the execution structure for the solution policy π' returned by* ND-Λ, *x the maximum number of state-action pairs that are added to any policy after* ND-Λ *visits a dead-end state, y is the maximum number of actions applicable to any state, and in every state s, $0 \leq d \leq y$ is the maximum number of actions applicable to s that lead to a dead-end state.*

Corollary 2 *Under the conditions of Theorem 2, if the number of possible successors of each state is bounded by a constant, then* ND-Λ *finds solutions in average time $O(\rho(|\pi'|) + \frac{xd|\pi'|}{y}))$, where $|\pi'|$ is the size of the solution.*

Experiments and Complexity Analysis

In order to perform an empirical evaluation of our theoretical results, we implemented ND-SHOP2, the nondeterminized version of SHOP2. We compared its performance and scalability for reachability goals with MBP (Bertoli *et al.* 2001), a well-known planning system for nondeterministic domains that is based on symbolic model checking techniques. In our comparisons, we used MBP's *Local Search* algorithm for finding strong-cyclic plans.[1] All of our experiments were run on an AMD Duron 900Mhz laptop computer running Linux RedHat 7.2 with 256MB memory.

Our first experimental domains was the Robot Navigation domain that was used for experimental evaluation of MBP in (Pistore, Bettin, & Traverso 2001). This domain is a variant of a similar domain described in (Kabanza, Barbeau, & St-Denis 1997). It consists of a building with 8 rooms connected by 7 doors. In the building, there is a robot and there are a number of packages in various rooms. The robot is responsible for delivering packages from their initial locations to their final locations by opening and closing doors, moving between rooms, and picking up and putting down the packages. The robot can hold at most one package at any time. To add nondeterminism, the domain also involves a "kid" that can close any of the open doors that are designated initially as "kid-doors."

We compared ND-SHOP2 and MBP with the same set of experimental parameters as in (Pistore, Bettin, & Traverso 2001): the number of packages n ranged from 1 to 5, and the number of kid-doors k ranged from 0 to 7. For each combination of n and k, we generated 20 random problems, ran the planners on the problems, and averaged the CPU time for each planner.[2]

Figure 3 shows the results for $k = 7$ and $n = 1, \ldots, 5$; this illustrates the behavior of the algorithms as the size of the domain increases.

Our experimental results confirm the ones in (Pistore, Bettin, & Traverso 2001): in both their experiments and ours, MBP's CPU time grows exponentially (the logarithm of the data grows linearly) in the size

[1] We also tried using MBP's Global Search algorithm, but the Local Search algorithm ran faster.

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

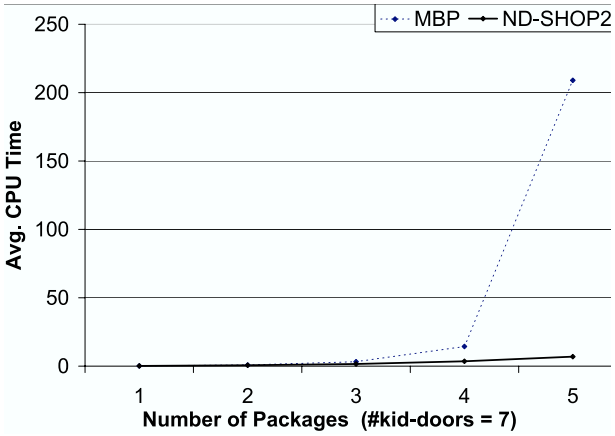

Figure 3: Average running times of ND-SHOP2 and MBP on Robot-Navigation problems, as a function of the number of packages. The number of kid-doors in the domain is fixed to 7.

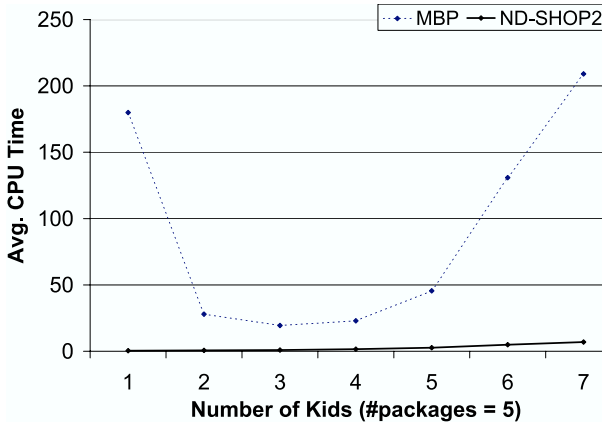

Figure 4: Average running times of ND-SHOP2 and MBP on Robot-Navigation problems, as a function of the number of kid-doors. The number of packages in the domain is fixed to 5.

of the problem. In contrast, the data for ND-SHOP2 show its CPU time growing polynomially, at only about $\Theta(n^4)$. Using the results in the previous section, we have confirmed that this is the correct growth rate for ND-SHOP2 in this domain.

To illustrate the scalability of the algorithms as the amount of nondeterminism increases, Figure 4 shows the results for $n = 5$ and $k = 1, \ldots, 7$. In each case, MBP takes one to two orders of magnitude more time than ND-SHOP2. The closest runtimes occurred at $k = 4$, where MBP is about 15 times slower than ND-SHOP2.

The reason for ND-SHOP2's fast performance relative to MBP is as follows. In the representation language for the Robot Navigation domain, a policy can say things along the lines of "if we are at door number 3 and it is open, then go through it," rather than having to give explicitly all of the exponentially many states of the world in which we're at door number 3 and the door is open. Because of this, problems in the robot-navigation domain have strong-cyclic solutions of linear size, and these are the solutions that ND-SHOP2 finds. Although MBP represents policies in a similar way, it apparently does not exploit this representation well enough to produce policies of polynomial size.

We also compared ND-SHOP2 and MBP on a nondeterministic version of the classical Blocks World domain. We introduced nondeterminism by allowing the operators to have three kinds of outcomes: (1) their traditional effects, (2) an operator may fail to change the state at all, and (3) an operator may fail by dropping the block onto the table.

Figure 5 shows the results of our experiments. Like before, each data point is the average of 20 random problems. For MBP, there are no data points for $n > 8$ because it was unable to solve any problems within the alloted time (30 minutes per problem). The logarithm of MBP's CPU time is linear; thus on this problem, like the other one, MBP takes exponential time.

Curve-fitting on ND-SHOP2's running time shows it growing at only about $\Theta(n^5)$, and a complexity analysis confirms this polynomial behaviour. The reasons are similar to the ones earlier: each time an operator puts a block in the wrong place, ND-SHOP2's methods tell it to pick it up and try again. As a result, ND-SHOP2 is guaranteed to produce a policy of size $O(n)$ where n is the number of blocks. In contrast, MBP produces exponential-size policies that tell what to do in most of the states of the world.

It would also be possible to write nondeterministic versions of the blocks world with more complicated kinds of nondeterminism: for example, an operator could drop a block not just onto the table, but onto any clear block. We have not yet tested this case experimentally, but our complexity analysis shows that such an experimental study would yield results similar to the above. ND-SHOP2 would take polynomial time and space, although the space would this time be quadratic rather than linear (it would immediately pick up the fallen block again, but in the worst case there would be $O(n)$ different places to pick up this block from). MBP would take exponential time in the size of the problem, for the same reasons as before.

Although we have not implemented our pseudocode for ND-TLPlan, the nondeterminization of TLPlan, we have analyzed its complexity on the planning domains discussed in this section. The analysis shows that with appropriate control rules, it would perform similarly to ND-SHOP2 in these domains. The details of our complexity analyses can be found in (Kuter & Nau 2004).

Related Work

One of the first approaches for planning in nondeterministic domains is the idea of *conditional planning*, which uses an extended form of STRIPS operators to have mutually-exclusive and conditional effects (Pryor & Collins 1996; Peot & Smith 1992). This approach

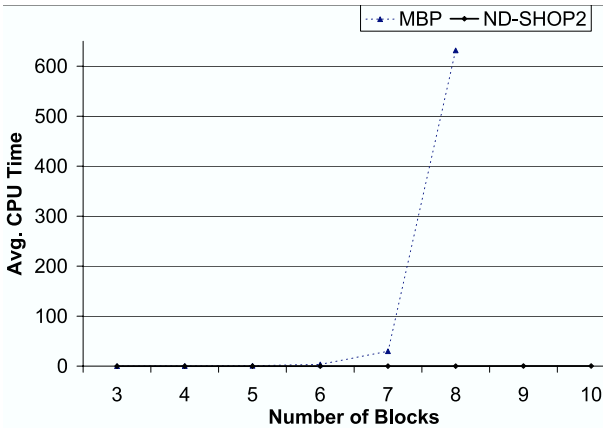

Figure 5: Average running times of ND-SHOP2 and MBP in the nondeterministic Blocks-World domain, as a function of the number of blocks.

does not scale up for complex planning domains.

The best-known approach for planning in nondeterministic domains is based on *symbolic model-checking* techniques (Cimatti *et al.* 2003; Cimatti, Roveri, & Traverso 1998; Daniele, Traverso, & Vardi 1999; Pistore, Bettin, & Traverso 2001). These works establish the notions of weak, strong, and strong-cyclic solution and algorithms based on symbolic techniques for finding such solution to nondeterministic planning problems. MBP (Bertoli *et al.* 2001) is an implementation of the ideas presented in these works.

Planning based on *Markov Decision Processes (MDPs)* (Boutilier, Dean, & Hanks 1999) also has actions with more than one possible outcome, but models the possible outcomes using probabilities and utility functions, and formulates the planning problem as an optimization problem. In MDPs, a policy is a total function from states to actions, whereas model-checking approaches allow a policy to be partial. For problems that can be solved either by MDPs or by model-checking-based planners, the latter have been empirically shown to be more efficient (Bonet & Geffner 2001).

Satisfiability and planning-graph based approaches have been extended to do planning in nondeterministic domains. To the best of our knowledge, the nondeterministic satisfiability-based planners (Castellini, Giunchiglia, & Tacchella 2003; Ferraris & Giunchiglia 2000)are limited to *conformant planning*, where the planner has nondeterministic actions and no observability. The planning-graph based techniques can address conformant planning and a limited form of partial-observabilitiy (Smith & Weld 1998; Weld, Anderson, & Smith 1998).

In deterministic planning domains, a key property of forward-chaining planners is that they know the current state at all times. This allows them to incorporate very powerful pruning techniques— particularly in hand-tailorable planners, in which the pruning techniques can be domain-specific. The domain-specific knowledge can be encoded into a planner in different forms. For example, HSP (Bonet & Geffner 1999) uses a heuristic function that allows the planner to choose the best successor state that can be reached by applying an action to the current state. Another heuristic-based planner, FF (Hoffmann & Nebel 2001), computes the heuristic values over the states by running a variation of the GraphPlan algorithm (Blum & Furst 1997) on a *relaxed* instance of the input planning problems. TLPlan (Bacchus & Kabanza 2000) and TALplanner (Kvarnström & Doherty 2001) use different kinds of *temporal-logic formulas*. In SHOP2 (Nau *et al.* 2003), domain-specific knowledge is encoded as *task networks* which SHOP2 *decomposes* recursively until it reaches a task network in which all tasks can be directly executed.

Two planning approaches that can exploit search control in nondeterministic domains are RTDP (Bonet & Geffner 2000), which is based on greedy-search techniques, and SIMPLAN (Kabanza, Barbeau, & St-Denis 1997). SIMPLAN was developed for planning with reactive agents in nondeterministic domains for temporally-extended goals. We did not consider SIMPLAN for our experiments since to the best of our knowledge it cannot produce strong-cyclic solutions.

Conclusions

During the past few years, some very efficient forward-chaining planners have been developed for planning in deterministic domains; there are many cases in which these planners can run in polynomial time. The goal of this paper has been to translate these advances in efficiency over to nondeterministic planning domains.

We have presented a general technique for taking forward-chaining planners for deterministic domains (e.g., HSP, TLPlan, TALplanner, and SHOP2), and *nondeterminizing* them to work in nondeterministic domains. The nondeterminization technique preserves soundness and completeness.

There are significant classes of nondeterministic planning problems in which the number of possible states is exponential but there are solutions whose execution structures have polynomial size. Our theoretical results suggest that in such domains, nondeterminizations of efficient deterministic planners may be able to do well.

The theoretical results are confirmed by our experiments and complexity analyses on two different problem domains. One of of these (Robot Navigation) has been used in several previous studies of planning in nondeterministic environments. In both problem domains, our experimental data show MBP's running time growing exponentially and ND-SHOP2's growing only polynomially. A complexity analysis for ND-SHOP2 confirms the latter result, and a complexity analysis for ND-TLPlan shows that it would perform similarly to ND-SHOP2 on these problems.

In the near future, we intend to improve our implementation of ND-SHOP2: for example, its implementation of ND-FCP's reachability tests is rather naive and

can be done much more efficiently. We intend to test our approach on additional planning domains, extend it to deal with temporally extended goals, and perhaps to implement nondeterminizations of additional planners.

In the longer term, we hope to extend the theory of nondeterministic planning domains to encompass several of the kinds of problem features that can be handled by planners such as SHOP2 and TLPlan, such as numeric computations, axiomatic inference, and calls to external information sources and software packages. We also hope to develop a technique for translating planners for deterministic domains into planners for probabilistic domains.

Acknowledgments

This work was supported in part by the following grants, contracts, and awards: Air Force Research Laboratory F30602-00-2-0505, Army Research Laboratory DAAL0197K0135, and Naval Research Laboratory N00173021G005. The opinions expressed in this paper are those of authors and do not necessarily reflect the opinions of the funders.

References

Bacchus, F., and Kabanza, F. 2000. Using Temporal Logics to Express Search Control Knowledge for Planning. *Artificial Intelligence* 116(1-2):123–191.

Bertoli, P.; Cimatti, A.; Pistore, M.; Roveri, M.; and Traverso, P. 2001. MBP: A Model Based Planner. In *Proceedings of IJCAI Workshop on Planning under Uncertainty and Incomplete Information*, 93–97.

Blum, A. L., and Furst, M. L. 1997. Fast Planning Through Planning Graph Analysis. *Artificial Intelligence* 90(1-2):281–300.

Bonet, B., and Geffner, H. 1999. Planning as Heuristic Search: New Results. In *Proceedings of the European Conference on Planning (ECP)*, 360–372. Durham, UK: Springer-Verlag.

Bonet, B., and Geffner, H. 2000. Planning with Incomplete Information as Heuristic Search in Belief Space. In Chien, S.; Kambhampati, S.; and Knoblock, C., eds., *Proceedings of the International Conference on AI Planning Systems (AIPS)*, 52–61. Breckenridge, CO: AAAI Press.

Bonet, B., and Geffner, H. 2001. GPT: A tool for planning with uncertainty and partial information. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 82–87. Seattle, WA: Morgan Kaufmann.

Boutilier, C.; Dean, T. L.; and Hanks, S. 1999. Decision-Theoretic Planning: Structural Assumptions and Computational Leverage. *Journal of Artificial Intelligence Research* 11:1–94.

Castellini, C.; Giunchiglia, E.; and Tacchella, A. 2003. SAT-Based Planning in Complex Domains: Concurrency, Constraints and Nondeterminism. *Artificial Intelligence* 147(1–2):85–117. To appear.

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.

Cimatti, A.; Roveri, M.; and Traverso, P. 1998. Strong Planning in Non-Deterministic Domains via Model Checking. In *Proceedings of the International Conference on AI Planning Systems (AIPS)*, 36–43. Pittsburgh, PA: AAAI Press.

Daniele, M.; Traverso, P.; and Vardi, M. 1999. Strong Cyclic Planning Revisited. In *Proceedings of the European Conference on Planning (ECP)*, 35–48. Durham, UK: Springer.

Ferraris, P., and Giunchiglia, E. 2000. Planning as Satisfiability in Nondeterministic Domains. In *AAAI/IAAI Proceedings*, 748–753. Menlo Park, CA: AAAI Press.

Hoffmann, J., and Nebel, B. 2001. The FF Planning System: Fast Plan Generation Through Heuristic Search. *Journal of Artificial Intelligence Research* 14:253–302.

Kabanza, F.; Barbeau, M.; and St-Denis, R. 1997. Planning Control Rules for Reactive Agents. *Artificial Intelligence* 95(1):67–113.

Kuter, U., and Nau, D. 2004. Using Forward-Chaining Planning for Generating Weak, Strong, and Strong-Cyclic Solutions in Nondeterministic Domains. Technical Report CS-TR-4580, Dept. of Computer Science, University of Maryland.

Kvarnström, J., and Doherty, P. 2001. TALplanner: A Temporal Logic-based Forward-chaining Planner. *Annals of Mathematics and Articial Intelligence* 30:119–169.

Nau, D.; Au, T.-C.; Ilghami, O.; Kuter, U.; Murdock, W.; Wu, D.; and Yaman, F. 2003. SHOP2: An HTN Planning System. *Journal of Artificial Intelligence Research* 20:379–404.

Peot, M., and Smith, D. 1992. Conditional Nonlinear Planning. In *Proceedings of the International Conference on AI Planning Systems (AIPS)*, 189–197. College Park, MD: AAAI Press.

Pistore, M.; Bettin, R.; and Traverso, P. 2001. Symbolic Techniques for Planning with Extended Goals in Nondeterministic Domains. In *Proceedings of the European Conference on Planning (ECP)*. Toledo, ES: Springer Verlag.

Pryor, L., and Collins, G. 1996. Planning for Contingency: A Decision-based Approach. *Journal of Artificial Intelligence Research* 4:81–120.

Smith, D. E., and Weld, D. S. 1998. Conformant Graphplan. In *AAAI/IAAI Proceedings*, 889–896. Menlo Park, CA: AAAI Press.

Weld, D. S.; Anderson, C. R.; and Smith, D. E. 1998. Extending Graphplan to Handle Uncertainty and Sensing Actions. In *AAAI/IAAI Proceedings*, 897–904. Menlo Park, CA: AAAI Press.