Maintaining Focus: Overcoming Attention Deficit Disorder in Contingent Planning

Ron Alford, Ugur Kuter, Dana Nau, Elnatan Reisner
Department of Computer Science
and Institute for Advanced Computer Studies
University of Maryland,
College Park, MD 20742, USA
{ronwalf,ukuter,nau,elnatan}@cs.umd.edu

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

Abstract

In our experiments with four well-known systems for solving partially observable planning problems (Contingent-FF, MBP, PKS, and POND), we were greatly surprised to find that they could only solve problems with a small number of contingencies. Apparently they were repeatedly trying to solve many combinations of contingencies at once, thus unnecessarily using up huge amounts of time and space.

This difficulty can be alleviated if the planner can maintain focus on the contingency that it is currently trying to solve. We provide a way to accomplish this by incorporating focusing information directly into the planning domain's operators, without any need to modify the planning algorithm itself. This enables the above planners to solve larger problems and to solve them much more quickly.

We also provide a new planner, FOCUS, in which focusing information can be provided as a separate input. This provides even better performance by allowing the planner to utilize more extensive focusing information.

Introduction

This paper deals with extending classical planning to *partially observable* planning problems, i.e., problems where the world is only partially known at planning time, and observations or queries must be done during plan execution, in order to decide which actions should actually be executed.

In the published literature, such planning problems have also been called *contingent*, *conditional*, and *incomplete-information* planning problems. However, those terms have also been used to refer to other kinds of planning problems (e.g., problems that require a conformant solution because no observations can be done during plan execution, or problems in which the actions have nondeterministic or probabilistic outcomes). Hence in an effort to avoid ambiguity, we will use the term *Partially Observable Classical* planning problem, or POC problem for short.

The best-known POC planning systems include Contingent-FF (Hoffmann & Brafman 2005), MBP (Cimatti *et al.* 2003), PKS (Petrick & Bacchus 2002), and POND (Bryce, Kambhampati, & Smith 2006). We have performed experimental comparisons of these planners in four benchmark problems used for these planners: the Unix domain, the Medicate domain, the Rovers domain, and a POC version of the Robot Navigation domain. In our tests, we expanded the problems by increasing the number of objects: the number of files, patients, objectives, or packages, respectively.

To our great surprise, we discovered that none of the four planners could solve problems that involved more than a small number of objects. The difficulty appeared to be a combinatorial explosion in the number of contingencies. For example, if there are n files or packages whose locations are unknown, or n patients who have unknown infections, then the number of *combinations* of locations or infections is exponential in n.

Apparently the reason why this difficulty did not appear in the authors' original tests was that most of their test problems were simple enough that there was no way for a combinatorial explosion to occur.[1,2]

The subject of this paper is how to enable POC planners to overcome combinatorial explosion in the number of combinations of contingencies. We describe a general technique that makes systematic modifications to planning domain's operators to help a planner focus its search on whichever contingency the planner is currently trying to solve. In our experiments, this focusing information provided significant improvements in the planners' performance, and it did not require any modifications to the planners themselves.

We also describe a new POC planning algorithm, FOCUS, that can make use of much more extensive focusing information. FOCUS can make use of focusing information written in a format similar to the HTN methods used in SHOP2 (Nau *et al.* 2003). In our ex-

[1] For example, there was one file to be moved in the Unix domain, one patient in the Medicate domain, and one objective in the Rovers domain.

[2] The Robot Navigation domain used in the published tests of MBP has large combinations of a kind of contingency that MBP can easily handle. Our version of the domain contained a different kind of contingency that caused MBP much more difficulty. For more details, see our "More Extensive Focusing" section.

periments, this focusing information enabled FOCUS to solve all of our planning problems very rapidly.

Formalism

Our definition of a POC planning domain is a generalization of that of a classical planning domain (e.g., (Ghallab, Nau, & Traverso 2004, Chapter 2)).

We define a *belief state* as a set b of ground literals. b contains all ground literals that are currently known to be true. If A is the set of all possible atoms, then the set of *unknowns* is $U = A - b - neg(b)$, where $neg(b)$ is the set of all literals that are negations of literals in b. A *completion* of b is any classical state (i.e., set of ground atoms) that contains all of the positive literals in b and none of the atoms whose negations are in b. $K(b)$ is the set of all completions of b.[3]

A planning operator o has preconditions $pre(o)$ and effects $eff(o)$, both of which are sets of literals. An action α is any ground instance of o. α is *applicable* in a belief state b if $pre(\alpha) \subseteq b$. If α is applicable, the result of applying it is $\gamma(b, \alpha) = \{b - neg(eff(\alpha))\} \cup eff(\alpha)$, where $neg(eff(\alpha))$ is the set of all literals that are negations of the literals in $eff(a)$.

There is an action $observe(p)$, for observing (at execution time) the ground atom p's current truth value. If p is true and currently observable, $observe(p)$'s effect is p. If p is false and currently observable, then $observe(p)$'s effect is $\neg p$. If p is not currently observable then $observe(p)$ has no effect. Thus, the result of applying $observe(p)$ in a belief state b is $\gamma(b, observe(p)) = \{(b \cup \{p\}), (b \cup \{\neg p\})\}$.

A *POC* planning problem is a 4-tuple $P = (O, N, b_0, g)$, where O is the set of operators, N is the set of sensing actions, b_0 is the *initial* belief state, and g is the goal. A *completion* of P is any classical planning problem $P' = (O, s_0, g)$ such that s_0 is a completion of b_0. $K(P)$ is the set of all completions of P.

A *policy* is a set of pairs $\pi = \{(b_1, a_1), \ldots, (b_n, a_n)\}$, where each b_i is a belief state, each a_i is an action, and $K(b_i) \cap K(b_j) = \emptyset$ whenever $i \neq j$. π's *execution structure* is a digraph Σ_π in which the nodes are all of the belief states that can be reached by applying actions in π, and the edges are the state transitions for those actions. If there's a path in Σ_π from b_1 to b_2, then b_1 is a π-ancestor of b_2 and b_2 is a π-descendant of b_1. A *leaf* in Σ_π is a belief state for which π does not specify any actions. A *dead-end* is a belief state b if there are no applicable actions in b or b does not have any leaf π-descendants.

A policy π is a *solution* of a POC planning problem P if every node in Σ_π is a π-ancestor of at least one goal node and every leaf node satisfies the goals of P.

Theorem 1 *π is a solution for P iff π is a solution for every completion of P.*

[3]Note that the number of the states in $K(b)$ is combinatorial in the number of the ground atoms in the planning domain since for each atom a in U, there will be two states in $K(b)$: a will be true in one and false in the other.

Table 1: CPU times in seconds for Contingent-FF, MBP, PKS, and POND. Each data point is the average of 100 randomly generated problems. For Contingent-FF we used the"enforced observations" option reported in (Hoffmann & Brafman 2005). Missing data points indicate that the planner either ran out of memory or exceeded our 2-hour time limit.

(a) The Unix domain

No. of files:	1	2	3	4	5
Contingent-FF	0.02	0.98	24.99	—	—
MBP	1.08	—	—	—	—
PKS	—	—	—	—	—
POND	0.37	—	—	—	—

(b) The Robot Navigation Domain

No. of packages:	1	2	3	4	5
Contingent-FF	0.02	0.33	5.57	—	—
MBP	0.08	—	—	—	—
PKS	1.32	265.95	—	—	—
POND	—	—	—	—	—

(c) The Medicate Domain

No. of patients:	1	2	3	4	5
Contingent-FF	0.01	0.13	1.87	18.25	—
PKS	1.32	265.95	—	—	—
POND	0.00	—	—	—	—

(d) The Rovers Domain

No. of objectives:	1	2	3	4	5
Contingent-FF	0.01	0.20	3.17	—	—
PKS	1.32	265.95	—	—	—
POND	0.14	—	—	—	—

Baseline Experiments

We compared Contingent-FF, MBP, PKS, and POND experimentally in the following well-known planning domains that we modified for POC planning: Unix (Petrick & Bacchus 2002), Robot Navigation (Cimatti *et al.* 2003; Kabanza, Barbeau, & St-Denis 1997), Medicate, (Petrick & Bacchus 2002; Hoffmann & Brafman 2005), and Rovers (Fox & Long 2002; Hoffmann & Brafman 2005; Bryce, Kambhampati, & Smith 2006).

The experiments were run on quad-core Xeon processors running at a clock speed of 2.33 Ghz. Note that while we used a multiprocessor system to run our experiments in parallel, none of the planners in our experiments used more than a single processor.

Table 1 shows the results of the experiments. None of the planners could solve problems in which there were more than a small number of unknowns (i.e., file locations, package locations, infections, or waypoints at which scientific objectives can be achieved). Contingent-FF's "enforced-observations" option, which enforces the observation actions as soon as they are applicable (otherwise, Contingent-FF's heuristic seemed to be ignoring them at their proper places in the plan,

yielding huge amount of search), enabled it to do better than the other planners, but Contingent-FF still could handle only a few unknowns. We have not run MBP on Medicate and Rovers because it was not able to solve these problems at all in the results reported in (Hoffmann & Brafman 2005).

Maintaining Focus

We believe the poor performance results in Table 1 derive from a kind of "attention deficit disorder" in the planners, which occurs due to a combinatorial explosion in the number of combinations of contingencies.

For example, in the Robot Navigation domain, if there are n packages and 7 rooms, then there are 7^n possible combinations of initial locations for those packages. If the planner cannot maintain focus on one contingency at a time, then it can waste huge amounts of time and space generating an alternative plan for each combination of contingencies. This leads us to the following hypothesis: *all four planners should be able to solve POC planning problems more efficiently if we can instruct them to focus only on a small number of contingencies at a time, rather than trying to consider all of them at once.*

In this section we'll consider a specific kind of focusing: that is, *focusing on the actions relevant for a specific task.* For example, if the planner can focus on delivering one package at a time, then there are only 7 possible locations for this package, hence 7 contingencies to consider. The planning should go more efficiently if the planner finishes planning for this package before considering the possible locations of any other package.

It is possible to implement this kind of focusing by rewriting the planning operators for a domain, without any need to modify the planners themselves. The technique is basically as follows: (1) Allow the current state to contain an assertion telling what subproblem a planner is currently trying to solve, and introduce a "focusing" action to put this assertion into the current state; (2) include this assertion in the preconditions of the operators for the subproblem; and (3) in the final action needed to complete the subproblem, include an effect that removes the "focusing" assertion, to tell the planner it can go on to some other subproblem.

As an example, here is an operator to assert that we are focusing on a package x in Robot Navigation:

focus(x)
  Precond: ¬focusing()
  Effects: focusing(), focused-on(x)

We will include focusing() in the preconditions of all of the robot's movement operators, and include focused-on(x) in the preconditions of the operators for picking up x and putting it down. In addition, we will modify the operator for putting down x so that its effects include ¬focusing() and ¬focused-on(x).

Furthermore, the operator for putting down a package x is useful only when the robot is at x's destination. Hence, to focus the planner even further, we'll add a precondition to this operator to prevent it from being used except when the robot is at x's destination.

It is easy to make similar modifications to the other domains. In the Unix domain, the focusing operator tells the planner to focus on one file at a time; in the Medicate domain, it tells the planner to focus on a particular patient; and so forth.

Table 2: CPU times for Contingent-FF, MBP, PKS, and POND. Each data point is the average of 100 randomly generated problems. Below, we used for Contingent-FF the"enforced observations" option reported in (Hoffmann & Brafman 2005). Missing data points indicate that the planner either ran out of memory or exceeded our 2-hour time limit.

(a) Unix domain, with focusing

Files:	1	2	3	4	5
Contingent FF	0.02	0.27	4.53	59.39	—
MBP	0.74	3.94	—	—	—
PKS	0.11	0.99	8.27	67.58	664.98
POND	0.15	—	—	—	—

(b) Robot Navigation domain, with focusing

Packages:	1	2	3	4	5
Contingent-FF	0.02	0.36	4.78	56.44	—
MBP	0.06	131.77	—	—	—
PKS	0.44	7.33	116.02	—	—
POND	—	—	—	—	—

(c) Medicate domain, with focusing

Patients:	1	2	3	4	5
Contingent-FF	0.00	0.05	0.58	3.95	—
PKS	0.00	0.06	0.38	1.73	10.13
POND	0.00	—	—	—	—

(d) Rovers domain, with focusing

Objectives:	1	2	3	4	5
Contingent-FF	0.00	0.05	0.58	3.95	—
PKS	0.00	0.14	1.32	10.76	81.15
POND	1.13	—	—	—	—

Experimental Results with Focusing. To test how well the above focusing information would help Contingent-FF, MBP, PKS, and POND, we ran them on the same planning problems as before, using the modified planning domains described above.

As shown in Table 2, most of the planners performed much better than before. Except for POND, they all could solve larger problems than before (especially PKS and Contingent-FF), and in most cases they ran much faster than before.

More Extensive Focusing

The previous section described one kind of focusing, implemented as modifications to the planning domain's operators. Here are some other kinds of focusing that are more awkward to implement in that fashion:

- *Focusing on relevant actions at multiple levels of tasks, subtasks, sub-subtasks, etc.* This is a more complicated version of the kind of focusing discussed in the previous section.
- *Focusing on the information relevant for deciding on the current action.* When solving one task x, there may be many unknowns whose values are irrelevant for solving x. In developing a plan for x, we shouldn't plan separately for each possible combination of values of these unknowns, because our plan for x will be the same in every case. For example, if we are in room4, then the open/closed status of door 4 matters, but the open/closed status of doors 2, 3, 5, 6, and 7 doesn't: the same set of actions will be applicable in the current state, regardless of whether those doors are open or closed.
  MBP and POND already implement a version of this kind of focusing, by planning over sets of states—for example, the set of all states in which door 4 is open—rather than individual states. They use a special-purpose data structure called a Binary Decision Diagram (BDD) to represent a set of states.
- *Focusing on a specified task ordering.* In some cases, it may be useful to impose an ordering on tasks or actions. As one example, to deliver a package x in the Robot Navigation Domain, we should find x, pick it up, move to x's destination, and put x down; and in the Unix domain, there is a similar ordering on the actions for finding and moving a file. As another example, rather than considering all 24 sequences in which we could deliver n packages, we might want to impose an arbitrary ordering such as package1, package2, package3, package4.

The FOCUS Algorithm. To enable advanced kinds of focusing such as the ones discussed above, we decided that instead of trying to make modifications to the planning operators, it would be easier to write the focusing information as Hierarchical Task Network (HTN) methods. To accomplish this, we wrote a new planner called FOCUS[4] that can utilize focusing information written as HTN methods. The pseudocode for FOCUS shown in Figure 1. FOCUS's HTN methods are written in a format similar to the format of SHOP2's methods (Nau *et al.* 2003), but generalized for use in POC domains.

FOCUS's input consists of an initial belief state b_0, a set O of planning operators, a set N of observation actions. The goals of the planning problem are expressed as tasks. FOCUS searches over a space of belief states. In order to plan for the various possible states in each belief state, it essentially does an AND-OR search (Nilsson 1980) in which each OR-branch corresponds to the choice of alternative methods for nonprimitive tasks, and each AND branch (which is represented by the set B in Figure 1) corresponds to the possible outcomes of the *observe* actions. In order to solve B, FOCUS must solve all of task networks that appear in B. FOCUS terminates when all of the task networks in B are empty; at that point, the policy π is a solution to the POC planning problem and FOCUS returns it.

[4]FOCUS is an acronym for "Focus On Contingencies Until Solved."

Procedure FOCUS(O, N, b_0, w_0, M)
 $\pi \leftarrow \emptyset$; $B \leftarrow \{(b_0, w_0)\}$;
 loop
  remove any pair (b, w) from B s.t. w is the empty HTN
  if $B = \emptyset$ then return(π)
  select a pair $(b, w) \in B$ and remove it from B
  if b does not appear in π then
   select a task t from w that has no predecessors
   if t is a primitive task then
    t corresponds to an action a
    $B \leftarrow$ ReFocus(B, b, w, t, a)
    if $B = failure$ then return $failure$
   else if t is a nonprimitive task then
    nondeterministically choose a method $m \in M$ for t
    $B \leftarrow$ ReFocus(B, b, w, t, m)
    if $B = failure$ then return $failure$
  else if b is dead-end in π then return($failure$)

Procedure ReFocus(B, b, w, t, x)
 if x an action for task t that is applicable in b then
  insert (b, x) into π
  remove t from w and insert $(\gamma(b, t), w)$ into B
 if x an HTN method for t that is applicable in b then
  remove t from w
  insert into w the decomposition of t specified in m
  insert (b, w) in B
 else if x has precondition $p \notin b$ and $observe(p) \in N$ then
  insert $(b, observe(p))$ into π
  $B \leftarrow B \cup \{(b', w) \mid b' \in \gamma(b, observe(p))\}$
 else return $failure$

Figure 1: Pseudocode for FOCUS. In the above, b_0 is the initial belief state, w_0 is the initial HTN task network, and M is the set of HTN methods.

FOCUS's task-decomposition process (the ReFocus subroutine in Figure 1) is similar to SHOP2's, but differs in several significant respects:

- Let *pre* be the preconditions of the operator or method x. In FOCUS, x is applicable in a belief state b if $pre \subseteq b$, that is, if each precondition $p \in pre$ is known to be true in b.
- If *pre* contains a precondition p whose truth value is unknown in b, then FOCUS checks whether there is an action $observe(p)$. If there is, then FOCUS generates two successor belief states: b' for the case where p is true, and b'' for the case where p is false; and it inserts the new pairs (b', w) and (b'', w) into B. Note that FOCUS does not modify the current HTN w; this won't happen until later in one of FOCUS's subsequent loop iterations.
- If p is false in b, or if p's truth value is unknown in b and there is no observation action for p, then FOCUS returns failure.

Table 3: CPU times in seconds for FOCUS, as a function of the number of files, packages or patients.

Files or packages:	1	2	3	4	5
Unix:	0.01	0.02	0.04	4.23	14.47
Robot Nav.:	0.57	1.64	3.09	5.26	8.15
Medicate:	0.00	0.03	0.06	0.11	0.20

Experimental Results. We tested FOCUS's performance on the Unix, Robot Navigation, and Medicate domains, [5] under the same experimental conditions described earlier. For each domain, we wrote an HTN description of some ways to focus in that domain.[6]

As an example of the kind of information encoded in these HTN descriptions, consider the Robot Navigation domain. For the task of delivering a package x, the relevant operations include the subtask of finding x, the operation of picking x up, the subtask of moving to x's destination, and the operation of putting x down. All of that can be easily specified as a single HTN method. For the subtask of moving to x's destination, the relevant operations include opening the door of the current room, moving to the hallway, opening the door of the destination room, and moving into that room. This can easily be specified as another HTN method.

The HTN representation made it possible to provide focusing information for each task without needing to encode additional preconditions and effects into the operators. Furthermore, it made for very efficient planning. As shown in Table 3, FOCUS very quickly solved all of the problems in our test suites.

Related Work

One of the earliest POC planning algorithm is CNLP (Peot & Smith 1992), a partial-order causal-link planner (a variant of SNLP) that generates conditional plans. Cassandra (Pryor & Collins 1996) was another partial-order planner contemporaneous with CNLP. Plinth (Goldman & Boddy 1994), differed from Cassandra and CNLP in being a linear (total-order) planner; this made it considerably simpler, and avoided CNLP's incompleteness issues.

In this paper, we considered four state-of-the-art planners that can deal with partial-observability in planning domains. PKS is a simple forward-chaining planning algorithm capable of performing depth-first or breadth-first search over a space of "knowledge" states. Each such state specifies a particular type of knowledge for planning, including facts known at planning time and facts whose truth value will be known at execution time. PKS generates conditional plans with branching points based on the knowledge for what is true/false at planning time and what will be true/false at execution.

MBP is probably the best-known planner for nondeterministic environments. It incorporates several algorithms based on symbolic model-checking (Cimatti *et al.* 2003), including algorithms to to deal with partial observability (Bertoli *et al.* 2006) in nondeterministic domains. In MBP, belief states are defined as sets of states that represent common observations, and compactly represented using Binary Decision Diagrams (BDDs) (Bryant 1992).

Contingent-FF (Hoffmann & Brafman 2005) is a POC planner that is an extension of the well-known FF planner (Hoffmann & Nebel 2001). Contingent-FF represents belief states implicitly, as partial sequences of actions, as oppossed to an explicit representation as in our formalism here. Contingent-FF conducts its planning as an AND-OR planning process, similar to FOCUS. The difference is the AND-OR search (Nilsson 1980) is guided by FF-style domain-independent heuristics that are computed based on relaxations of POC planning problems.

POND (Bryce, Kambhampati, & Smith 2006) is also based on AO* search as Contingent-FF. POND includes some complicated planning-graph based heuristics for planning over belief states that are implemented via BDDs. Earlier experiments with POND demonstrated that it outperforms Contingent-FF and MBP in some planning benchmarks. In our planning domains, however, both Contingent-FF and MBP performed better than POND and solved larger problems. One possible explanation is that the heuristic selection of the successor states and actions in our domains does not conform well with the BDD-based state and policy representations in POND, which usually require sets of states be described via well structured logical formulas (Cimatti *et al.* 2003; Kuter & Nau 2004).

The C-SHOP planner described in (Bouguerra & Karlsson 2004) is an earlier investigation into using HTN planning in partially-observable domains. However, C-SHOP is based on the SHOP planner (Nau *et al.* 1999); hence unlike FOCUS it cannot interleave subtasks. Also, C-SHOP models observations over belief states via special tasks in an HTN that need to be provided by the domain expert and must include domain knowledge about probabilities and observation tasks. FOCUS does not require such knowledge: its HTN methods only contain information about how to focus its attention.

Finally, the work reported in (Baier, Fritz, & McIlraith 2007) describes a way to take a body of procedural domain knowledge specified in an Algol-like language and translates it into PDDL so that any classical planner that uses PDDL as input planning language is able to use the translated domain knowledge. We believe this work nicely dovetails with our notion of using focus operators, but it does not address POC plannning, as we do. It would be interesting as a future work to use this approach for generating domain knowledge for

[5] Based on our experience with the other three domains, we expect that FOCUS will do quite well on the Rovers domain, but we have not yet had time to carry out those experiments. We intend to do so in the near future.

[6] Our HTN domain descriptions are posted at http://www.cs.umd.edu/users/ronwalf/2009/focus/.

our framework described here.

Conclusions

For ordinary classical planning, planning graphs provided a very powerful technique for focusing a planner on a particular part of the search space. The planning graph provides a collection of operators, and ordering constraints on those operators, such that if a solution exists at all, a solution can be found using those operators with those ordering constraints.

For POC planning, a similar degree of focusing power is harder to attain. In our initial set of experiments with Contingent-FF, MBP, PKS, and POND—planners that represent the state-of-the-art in POC planning—none of them was able to solve planning problems with more than a few objects, because they were unable to focus on just a few of them at a time.

We have shown how some elementary focusing information can be encoded directly into POC planning operators, without making any modifications to the planning algorithms. As shown in our second set of experiments, this enabled Contingent-FF, MBP, and PKS to solve problems more quickly and solve bigger problems than they could solve before. The focusing information, however, did not improved the performance of POND since, we believe, the focusing information we mentioned above did not have any affect in constraining BDD representations in POND.

In addition, we have described a new planner called FOCUS, that makes use of focusing information written in an HTN-like language. FOCUS very quickly solved even our largest test problems.

As we explained earlier, the focusing information used by FOCUS is written in an HTN-like language because it would have been awkward to encode it directly into ordinary POC planning operators. On the other hand, we believe it will be possible to develop an algorithm to take the same focusing information used by FOCUS, and translate it automatically into POC planning operators. Developing such an algorithm will be a focus (please excuse the pun) of our near-term work. This will provide a way to incorporate FOCUS's focusing information into Contingent-FF, MBP, PKS, POND, and any other POC planning algorithm.

In the future, we intend to explore ways to automatically extract focusing information from the execution traces of solutions to planning problems in a domain. Currently, either in the form of focus operators or in more expressive HTNs for the FOCUS algorithm, this knowledge is hand-written by the domain experts. In addition to the idea mentioned in the previous section, another way we intend to investigate is to use goal-regression techniques to extract focusing knowledge given execution traces for planning problems.

Acknowledgments. This work was supported in part by AFOSR grant FA95500610405, NAVAIR contract N6133906C0149, DARPA's Transfer Learning and Integrated Learning programs, and NSF grant IIS0412812. The opinions in this paper are those of the authors and do not necessarily reflect the opinions of the funders.

References

Baier, J.; Fritz, C.; and McIlraith, S. 2007. Exploiting procedural domain-specific control knowledge in state-of-the-art planners. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*.

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

Bouguerra, A., and Karlsson, L. 2004. Hierarchical task planning under uncertainty. In *3rd Italian Workshop on Planning and Scheduling (AI*IA-04)*.

Bryant, R. E. 1992. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys* 24(3):293–318.

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.

Fox, M., and Long, D. 2002. International planning competition. http://www.dur.ac.uk/d.p.long/competition.html.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: Theory and Practice*. Morgan Kaufmann.

Goldman, R. P., and Boddy, M. S. 1994. Conditional linear planning. In *AIPS-94*, 80–85.

Hoffmann, J., and Brafman, R. 2005. Contingent Planning via Heuristic Forward Search with Implicit Belief States. In *Proceedings of the 15th International Conference on Automated Planning and Scheduling*.

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. Forward-chaining planning in nondeterministic domains. In *AAAI-2004*.

Nau, D. S.; Cao, Y.; Lotem, A.; and Muñoz-Avila, H. 1999. SHOP: Simple hierarchical ordered planner. In *IJCAI-99*.

Nau, D.; Au, T.-C.; Ilghami, O.; Kuter, U.; Murdock, W.; Wu, D.; and Yaman, F. 2003. SHOP2: An HTN planning system. *JAIR* 20:379–404.

Nilsson, N. 1980. *Principles of Artificial Intelligence*. Morgan Kaufmann.

Peot, M., and Smith, D. 1992. Conditional nonlinear planning. In *AIPS-92*, 189–197.

Petrick, R., and Bacchus, F. 2002. A knowledge-based approach to planning with incomplete information and sensing. In *AIPS-02*.

Pryor, L., and Collins, G. 1996. Planning for contingencies: A decision-based approach. *JAIR* 4:287–339.