Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-2001), Seattle, August 2001

Total-Order Planning with Partially Ordered Subtasks

Dana Nau
Univ. of Maryland, College Park, MD, USA
nau@cs.umd.edu

Héctor Muñoz-Avila
Univ. of Maryland, College Park, MD, USA
munoz @cs.umd.edu

Yue Cao
Solers Inc.
Arlington, VA
JasonC@homemail.com

Amnon Lotem
Estimotion, Ltd.
Glil Yam, Israel
lotem@cs.umd.edu

Steven Mitchell
Lockheed Martin
Manassas, VA, USA
steve.mitchell@lmco.com

Abstract

One of the more controversial recent planning algorithms is the SHOP algorithm, an HTN planning algorithm that plans for tasks in the same order that they are to be executed. SHOP can use domain-dependent knowledge to generate plans very quickly, but it can be difficult to write good knowledge bases for SHOP.

Our hypothesis is that this difficulty is because SHOP's total-ordering requirement for the subtasks of its methods is more restrictive than it needs to be. To examine this hypothesis, we have developed a new HTN planning algorithm called SHOP2. Like SHOP, SHOP2 is sound and complete, and it constructs plans in the same order that they will later be executed. But unlike SHOP, SHOP2 allows the subtasks of each method to be partially ordered.

Our experimental results suggest that in some problem domains, the difficulty of writing SHOP knowledge bases derives from SHOP's total-ordering requirement—and that in such cases, SHOP2 can plan as efficiently as SHOP using knowledge bases simpler than those needed by SHOP.

1 Introduction

One of the more controversial recent planning systems is the SHOP system [Nau *et al.*, 1999], an HTN planning system that plans for tasks in the same order that they will later be executed. Like any HTN planner, SHOP uses domain knowledge in order to plan more efficiently—but unlike other HTN planners, SHOP always generates the steps of its plans in the same order that those steps will later be executed.

On one hand, the SHOP algorithm makes it possible to generate plans quite efficiently. For example, in the experiments reported in [Nau *et al.*, 1999, 2000], SHOP ran orders of magnitude faster than the Blackbox, IPP, Tlplan, and UMCP planners. Furthermore, the SHOP algorithm is suitable for use as an embedded planning sytem in complex applications [Munoz *et al.*, 2000].

On the other hand, creating a SHOP knowledge base can require significantly more "programming effort" than is needed for action-based planners. For example, in two of the planning domains in Track 2 of the AIPS-2000 planning competition, SHOP was disqualified because we did not finish debugging the knowledge bases in time.

We believe that although the total-order HTN-decomposition technique used in SHOP has some significant benefits, the SHOP planning algorithm provides too restrictive a way of achieving these benefits. In particular, SHOP requires the subtasks of each method to be totally ordered, which makes it impossible for SHOP to interleave subtasks of different tasks. In Section 2 we describe how this can complicate the job of the knowledge-base author by requiring him/her to introduce "global planning" instructions into SHOP's knowledge bases that would not otherwise be needed.

In this paper we introduce the SHOP2 planning algorithm, which has the following properties:

- Like SHOP, SHOP2 is a sound and complete HTN planning algorithm that generates the steps of each plan in the same order that those steps will later be executed. Thus, like SHOP, SHOP2 knows the current state at each step of the planning process.
- Unlike SHOP, SHOP2 allows each method to decompose into a partially ordered set of subtasks, and allows the creation of plans that interleave subtasks from different tasks.
- SHOP2 is upward-compatible with SHOP. Our SHOP2 implementation can run SHOP knowledge bases with only minor syntactical changes, and in fact runs them more efficiently than SHOP does.

We have done experimental comparisons of SHOP and SHOP2 in problem domains exemplifying situations (1) where domain-specific global-reasoning knowledge seems necessary for efficient plan-generation regardless of SHOP's total-ordering requirement, and (2) where such knowledge is necessitated only by SHOP's partial-ordering requirement. In the latter case, we could easily create a knowledge base much simpler than SHOP's, that enabled SHOP2 to create plans more efficiently than SHOP and with plan quality similar to SHOP's.

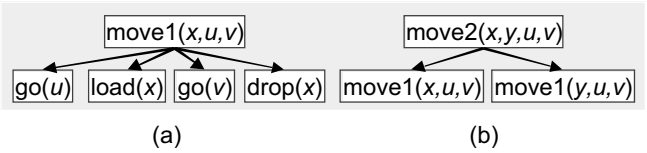

Figure 1. First set of methods for moving packages.

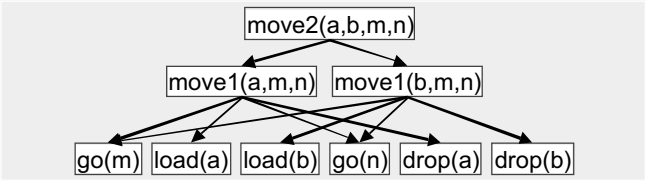

Figure 2: The plan that we want for two packages.

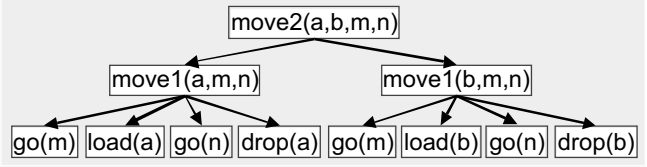

Figure 3: The plan we have actually told SHOP to generate.

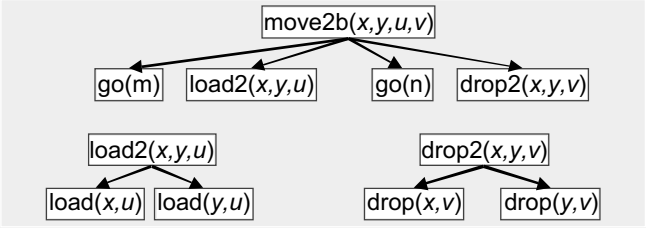

Figure 4. Second set of methods to move two packages.

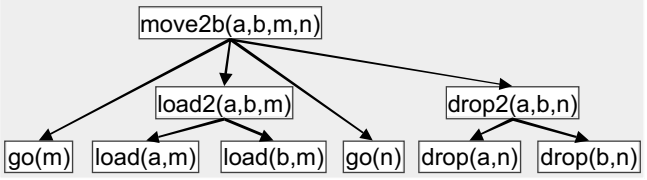

Figure 5. Plan generated using the methods in Figure 4.

2 Motivation

As example of the kind of difficulty that can result from SHOP's requirement that all tasks be totally ordered, consider the task of moving a package from one location to another. For SHOP to generate a plan for this task, it needs to have a method telling it how to move a package, such as the method shown in Figure 1(a). This method says that one way to move a package is to go to the package, pick the package up, go to the destination, and drop the package off.

If all we want to do is to deliver one package, then the method in Figure 1(a) will work fine. However, suppose we want to move two packages at once. Although the two methods shown in Figure 1 might at first glance seem satisfactory for this task, they will not always do what we want. If the two packages both have the same initial location and the same destination, then we probably would like to deliver both packages at once, as shown in Figure 2—but the methods in Figure 1 will tell SHOP to deliver the packages one at a time, as shown in Figure 3.

In this simple example, it is not hard to write methods telling SHOP to generate a plan for delivering both packages at once. For example, the methods shown in Figure 4 will tell SHOP to generate the plan shown in Figure 5. However, to do this we had to tell SHOP explicitly how to reason about both packages at once.

The need to give SHOP such "global planning" instructions can occur frequently. Each of the SHOP knowledge bases described in [Nau *et al.*, 1999] contains instructions for reasoning globally about the planning problem—and the same is true in most of the knowledge bases that we created during the AIPS-2000 planning competition. To write and debug such instructions can require significant time and effort.

3 SHOP2

3.1 Preliminaries

As with most AI planners, a logical atom in SHOP2 consists of a predicate name followed by a list of arguments, and a state of the world consists of a collection of ground logical atoms.

As with most HTN planners, a task in SHOP2 consists of a task name followed by a list of arguments. Although tasks are similar syntactically to logical atoms, they are different semantically since they denote activities that need to be performed rather than logical conditions [Erol *et al*, 1994; Barret, 1997].

A SHOP2 knowledge base contains domain-specific knowledge that SHOP2 will need in order to do planning in some domain. It consists of axioms, methods, and operators, as described below.[1]

Axioms. SHOP2's Horn-clause axioms are identical to those used in SHOP. As in SHOP, SHOP2 uses these axioms to infer whether a method's preconditions are satisfied in the current state of the world, using Horn-clause inference.

Also like SHOP, SHOP2's methods and Horn clauses can contain calls to the Lisp evaluator. This allows SHOP2 to evaluate preconditions that contain, for example, numeric computations or queries to external information sources.

Methods. SHOP2's methods are similar to those of SHOP except that methods can produce partially ordered sets of subtasks. As in SHOP, the basic form of a method is

[1] SHOP2 generalizes the M-SHOP algorithm described in [Nau *et al.*, 2000]. M-SHOP takes partially ordered task lists as input—but like SHOP, its methods must produce totally ordered sets of subtasks. In contrast, SHOP2 allows methods to produce partially ordered sets of subtasks.

(:method *task-atom precondition-atoms decomposition*)

The *task-atom* tells what kind of task the method can be used for. The method cannot be applied to some task *t* unless *task-atom* is unifiable with *t*).

The *precondition-atoms* tell what things must be true in the current state of the world in order for the method to be applicable to *t*. The previous paragraphs described how SHOP2 infers whether these preconditions are true in the current in the current state.

The *decomposition* tells what subtasks to decompose the task into. In SHOP this set of subtasks was totally ordered, but in SHOP2 it can be partially ordered.

As in SHOP, additional preconditions and decompositions can be appended to the method for SHOP2 to use in an "if-then-else" manner:

(:method
task-atom
precondition-atoms-1
decomposition-1
precondition-atoms-2
decomposition-2 ...)

The idea here is that if *precondition-atoms-1* is true then the method will produce *decomposition-1*; otherwise if *precondition-atoms-2* is true then the method will produce *decomposition2*, and so forth.

Operators. Since the subtasks of a method can be partially ordered, this means that subtasks of different methods can be interleaved in a plan. Thus in order to prevent deleted-condition interactions, we need a way to specify protected conditions. However, since SHOP2 will plans for tasks in the order that they will later be executed, we can get by with a rather simple protection mechanism, rather than the more sophisticated mechanisms used in partial-order HTN planners such as O-PLAN [Currie and Tate, 1991; Tate, 1994], SIPE [Wilkins, 1990], and UMCP [Erol *et al.*, 1994]. To accomplish this, SHOP2's operator syntax is modified beyond that of SHOP, to include the following way to specify protected conditions.

Like SHOP's operators, each SHOP2 operator includes a task atom (which must be unifiable with a task in order for the operator to be applicable to that task), a "delete list" (which tells what atoms to delete from the current state), and an "add list" (which tells what atoms to add to the current state). However, SHOP2 operators can also include *protection requests* (to tell SHOP2 that certain conditions should not be deleted) and *protection cancellations* (to tell SHOP2 that it is now permissible to delete those conditions).

For example, suppose we want to tell SHOP2 to drive a truck from location *p* to location *q*, and prevent the truck from being moved away from *q*. Then we would write an operator that deletes "(at-truck *p*)", adds "(at-truck *q*)", and adds a protection request for "(at-truck *q*)". Once we are ready to move the truck, another operator can delete the protection request.

3.2 The SHOP2 Algorithm

The SHOP2 planning algorithm is as follows, where *S* is the current state, *M* is a partially ordered set of tasks, and *L* is a list of protected conditions:

procedure SHOP2(*S*,*M*,*L*)
  if *M* is empty **then return** NIL **endif**
  nondeterministically choose a task *t* in *M* that has no predecessors
  <*r*,*R'*> = reduction(*S*,*t*)
  if *r* = FAIL **then return** FAIL **endif**
  nondeterministically choose an operator instance *o* applicable to *r* in *S*
  S' = the state produced from *S* by applying *o* to *r*
  L' = the protection list produced from *L* by applying *o* to *r*
  M' = the partially ordered set of tasks produced from *M* by replacing *t* with *R'*
  P = SHOP2(*S'*,*M'*,*L'*)
  return cons(*o*,*P*)
end SHOP2

procedure reduction(*S*,*t*)
  if *t* is a primitive task **then return** <*t*,NIL>
  else if no method is applicable to *t* in *S* **then return** <FAIL,NIL> **endif**
  nondeterministically let *m* be any method applicable to *t* in *S*
  R = the decomposition (partially ordered set of tasks) produced by *m* from *t*
  r = any task in *R* that has no predecessors
  <*r'*,*R'*> = reduction(*S*,*r*)
  if *r'* = FAIL **then return** <FAIL,NIL> **endif**
  R'' = the partially ordered set of tasks produced from *R* by replacing *r* with *R'*
  return <*r'*,*R''*>
end reduction

The proof that the SHOP2 algorithm is both sound and complete is too long to include in this paper. However, it is a relatively straightforward induction proof, proceeding from the usual kind of definition of what it means for something to be an HTN plan.

Probably the only complicating factor worth mentioning here is the reason why REDUCE calls itself recursively until it finds a primitive task. This is needed in order to ensure that for all of the methods used to produce that primitive task, the preconditions are evaluated in the correct state of the world.

4 Implementation and Experiments

We implemented the SHOP2 algorithm by modifying the Common-Lisp coding for the SHOP planning system. As we did with SHOP, we intend to make SHOP2 available as freeware under the GNU public license.

For our experiments, we wanted to compare SHOP2 and SHOP on problem domains exemplifying two different cases for the role of domain-dependent "global reasoning" knowledge:

- Cases where such knowledge is somehow an intrinsic requirement for generating plans efficiently. We chose the blocks world [Nilsson, 1980] as a problem domain where such cases might be likely to occur.
- Cases where such knowledge is necessitated only by SHOP's total ordering requirement. As a domain in which such cases would be likely, we chose the logistics domain [Veloso, 1992].

For our comparisons, we built knowledge bases for SHOP2 in the logistics and blocks-world domains, and compared them to the knowledge bases that come with the SHOP code. For our tests we used a 400-MHz Power Macintosh G4 with 256 MB of RAM, using Macintosh Common Lisp 4.3.

In order to do these comparisons properly, one concern was that in modifying the SHOP code to create the SHOP2 code, we also made some significant optimizations. Because of these optimizations, SHOP2 runs SHOP knowledge bases faster than SHOP does, so we felt that running the SHOP2 code against the SHOP code would be unfairly favorable to SHOP2.

To solve this problem, we utilized SHOP2's upward-compatibility with SHOP. With some minor syntactical changes, any SHOP knowledge base will run in SHOP2—and running such a knowledge base in SHOP2 is equivalent to running the SHOP planning algorithm. Thus for our tests, we used SHOP2 to run both the SHOP knowledge base and the SHOP2 knowledge base.

4.1 Logistics Problems

The SHOP knowledge base contains a complicated set of instructions that tell SHOP how to reason globally about the logistics-planning domain. We could not figure out a way to make any significant simplifications to this knowledge base and still have it run in SHOP, except by removing the domain knowledge and forcing SHOP to do a brute-force search. However, it was relatively easy for us to create a much simpler SHOP2 knowledge base, consisting of instructions for how to transport an individual package to its destination. As can be seen from Table 1, this resulted in a SHOP2 knowledge base whos size was only about 26% of the size of the SHOP knowledge base.

We compared the SHOP knowledge base with the SHOP2 knowledge base on 110 randomly generated logistics problems. The problems involved *N* packages to be delivered, for *N* = 10, 15, …, 60. There were 10 problems for each *N*, for a total of 110 problems. In each problem, the number of cities was no larger than N/2. Each city contained three locations, one truck, and *N*/5 or fewer airports. For each package, the original location and the destination location were randomly chosen and were guaranteed to be different from each other.

As shown in Figure 1, SHOP2 ran about 4 times as fast with the SHOP2 knowledge base as it did with the SHOP knowledge base. As shown in Figure 2, the two knowledge bases created plans of nearly the same size, but the ones generated by the SHOP knowledge base were slightly shorter.

Table 1: Sizes of the SHOP and SHOP2 knowledge bases for the logistics domain (counting each if-then decomposition of a method as a separate method).

	SHOP knowledge base	SHOP2 knowledge base
Methods	50	10
Operators	7	7
Axioms	10	1

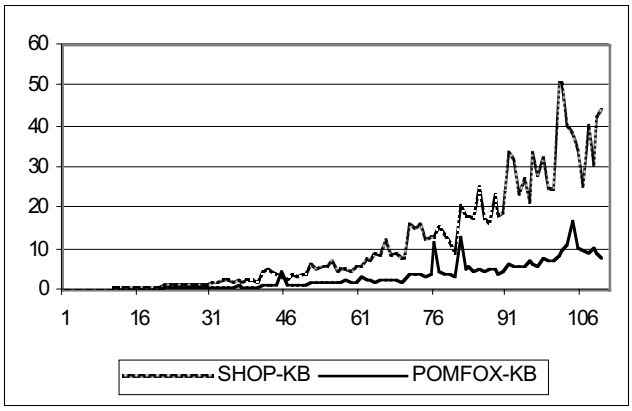

Figure 1. CPU times for SHOP2 using the SHOP knowledge base and the SHOP2 knowledge base, on 110 randomly generated logistics problems. The x-axis gives the problem number, and the y-axis gives the CPU time.

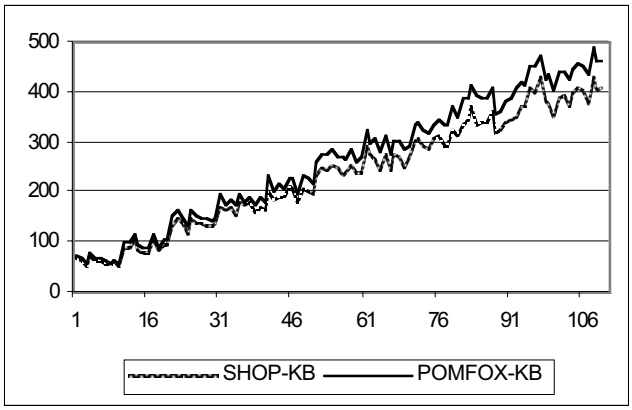

Figure 2. Sizes of the plans of Figure 1. The x-axis gives the problem number, and the y-axis gives the number of actions.

Table 2: Sizes of the SHOP and SHOP2 knowledge bases for the blocks world (counting each if-then decomposition of a method as a separate method).

	SHOP knowledge base	SHOP2 knowledge base
Methods	10	13
Operators	7	8
Axioms	1	5

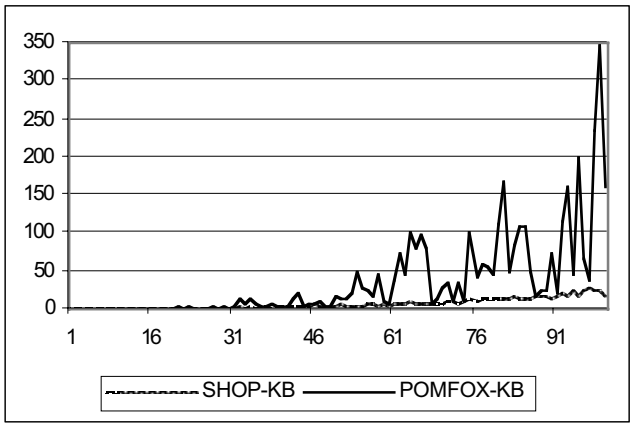

Figure 3. CPU times for SHOP2 using the SHOP knowledge base and the SHOP2 knowledge base, on randomly generated blocks-world problems. The x-axis gives the problem number, and the y-axis gives the CPU time.

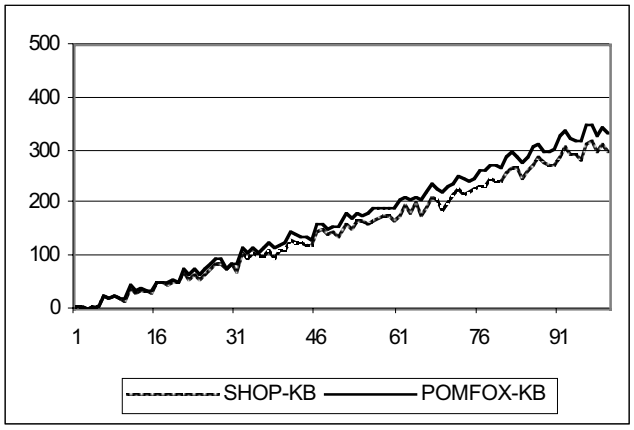

Figure 4. Sizes of the plans of Figure 3. The x-axis gives the problem number, and the y-axis gives the number of actions.

4.2 Blocks-World Problems

Just as before, the blocks-world knowledge base for SHOP contains instructions for how to reason globally about the planning process. Just as before, we could not think of any way to significantly reduce the size of this knowledge base and still have it run in SHOP, other than by removing the domain knowledge and forcing SHOP to do a brute-force search.

As before, our objective for SHOP2 was to simplify the knowledge base giving SHOP2 instructions for how to move individual blocks. However, we could not figure out any way to do this without forcing SHOP2 to do a brute-force search. In fact, we suspect that "global" domain-specific algorithms such as the ones discussed in [Chenoweth, 1991; Gupta and Nau, 1991] may be the only way to achieve efficient planning in the blocks world.

We tried creating a SHOP2 knowledge base by removing some of the total-ordering constraints and bookkeeping operations from the SHOP knowledge base, but this required us to add additional coding such as protection requests and protection cancellations to handle interleaving correctly. As shown in Table 2, the resulting knowledge base was about 44% larger than the original SHOP knowledge base.

We compared the performance of the two knowledge bases on randomly blocks-world problems consisting of N= 5,10,…,100 blocks to be relocated. We generated five problems for each value of N, for a total of 100 problems. To build the initial and final states, we generated configurations of blocks as follows:

- First, put a block onto the table (thereby creating a new tower).
- For each block after the first one, if t is the number of existing towers, then there are t+1 possible locations for the new block: on top of any of the existing towers, or on the table (thereby creating a new tower). Choose one of those locations at random, with an equal probability for each choice.

As shown in Figure 3, the time taken by SHOP2 using the SHOP2 knowledge base varied greatly from problem to problem. On the average, SHOP2 needed about 2.4 times as much time to generate plans with this knowledge base as it did with the SHOP knowledge base. As shown in Figure 4, the two knowledge bases created plans of nearly the same size, but the ones generated by the SHOP knowledge base were slightly shorter.

Discussion and Conclusions

We have described a new HTN planning algorithm, the SHOP2 algorithm. Like the SHOP planning algorithm, the SHOP2 algorithm generates the steps of a plan in the same order in which those steps are to be executed—but unlike SHOP, SHOP2 allows the subtasks of each method to be partially ordered.

SHOP2 runs SHOP knowledge bases faster than SHOP does; and our test results show that in some cases one can create knowledge bases for SHOP2 that are much simpler than the ones needed by SHOP yet still allow SHOP2 to run more quickly than SHOP.

We believe that the primary impact of our results is to provide a way to obtain the same advantages ascribed to the SHOP planning algorithm, while alleviating one of SHOP's primary drawbacks. Below, we summarize what those advantages and drawbacks are:

1. Planning for tasks in the order that those tasks will be performed makes it possible to know the current state of the world at each step in its planning process, which makes it possible to incorporate significant reasoning power into the planner's precondition-evaluation mechanism. Rather than just unifying preconditions against current-state atoms, the SHOP2 system (like the SHOP system) can perform Horn-clause inferences to evaluate preconditions that are not directly mentioned

in the current state, and its preconditions can incorporate calls to the Lisp evaluator (e.g., to do numeric computations or make queries to external sources of information).

2. The combination of HTN decomposition (to focus the search on the goal) and reasoning power in the preconditions (to prune inapplicable methods and operators from the search space) makes it possible to write domain-dependent knowledge bases that provide very efficient planning performance. As an illustration of what this means, when we tried to run Blackbox and IPP on the suites of logistics problems and blocks-world problems described above, we could not get them to solve any of the problems in the test suites. In each case, either they ran out of memory or else we had to terminate them after they had run for more than 30 minutes of CPU time without finding solutions.

3. The primary drawback of any HTN planning system is the effort needed to create a knowledge base of domain-dependent information for the domain we want it to do planning in. In SHOP, this drawback is sometimes worsened by SHOP's restriction that the subtasks of each method must be totally ordered, because this can require the knowledge-base author to introduce global reasoning into the planning domain that would not otherwise be needed. Our experimental results suggest that in these cases, SHOP2 can plan more efficiently than SHOP using knowledge bases much simpler than those needed by SHOP. In cases where such knowledge bases cannot be created, SHOP2 can run SHOP knowledge bases quicker than SHOP.

Some of our topics for future work include the following: investigating additional ways to make the SHOP2 algorithm more powerful and easier to use, releasing the coding for SHOP2 as open-source software, and using the SHOP2 algorithm as an embedded planning algorithm in a real-world application.

Acknowledgements

This work was supported in part by the following grants, contracts, and awards: AFRL F306029910013 and F30602-00-2-0505, ARL DAAL0197K0135, and U. of Maryland General Research Board. Opinions expressed here do not necessarily reflect those of the funders.

References

[Barrett, 1997] A. Barrett. *Frugal Hierarchical Task-Network Planning*. Computer Science and Engineering Dept., University of Washington, 1997.

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

[Chenoweth, 1991] S. V. Chenoweth. On the {NP}-hardness of blocks world. *AAAI-91*, July 1991, pages 623–628.

[Currie and Tate, 1991] K. Currie and A. Tate. O-Plan: The Open Planning Architecture. *Artificial Intelligence*, 52(1):49-86, 1991.

[Erol *et al.*, 1994] K. Erol, J. Hendler and D. S. Nau. UMCP: A Sound and Complete Procedure for Hierarchical Task-Network Planning. In *Proc. Second International Conf. on AI Planning Systems (AIPS-94)*, June, 1994, pages 249-254.

[Gupta and Nau, 199] N. Gupta and D. S. Nau. Complexity Results for Blocks-World Planning. *AAAI-91*, July 1991, pages 629-633.

[Kautz and Selman, 1999] H. Kautz and B. Selman. Unifying SAT-based and graph-based planning. *IJCAI-99*, 1999, pages 318-325.

[Koehler *et al.*, 1997] J. Koehler, B. Nebel, J. Hoffman and Y. Dimopoulus. Extending planning graphs to an ADL subset. In *Proc. ECP-97*, 1997. Toulouse, France.

[Munoz-Avila *et al.*, 2000] H. Munoz-Avila, D. W. Aha, L. A. Breslow, D. S. Nau and R. Weber. Integrating Conversational Case Retrieval with Generative Planning. In *EWCBR-2000*, 2000. Trento, Italy: Springer-Verlag.

[Nau *et al.*, 1999] D. Nau, Y. Cao, A. Lotem, and H. Muñoz-Avila. SHOP: Simple Hierarchical Ordered Planner. In *IJCAI-99*, 1999.

[Nau *et al.*, 2000] D. S. Nau, Y. Cao, A. Lotem, and H. Muñoz-Avila. SHOP and M-SHOP: Planning with Ordered Task Decomposition. Tech report TR 4157, University of Maryland, College Park, MD, June 2000.

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

[Tate, 1994] A. Tate. Mixed Initiative Planning in O-Plan2. In *Proceedings of the ARPA/Rome Laboratory Knowledge-Based Planing and Scheduling Initiative*, 1994, pages 512-516. Tuscon, AR: Morgan Kaufmann.

[Veloso, 1992] M. Veloso. "Learning by Analogical Reasoning in General Problem Solving." Tech. Report CMU-CS-92-174, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1992.

[Wilkins, 1990] D. Wilkins. "Can AI Planners Solve Practical Problems?" *Computational Intelligence*, 6(4):232-246, 1990.