Applications of SHOP and SHOP2

Dana Nau, Tsz-Chiu Au, Okhtay Ilghami, Ugur Kuter, Dan Wu, and Fusun Yaman, *University of Maryland*

Héctor Muñoz-Avila, *Lehigh University*

J. William Murdock, *IBM Research*

We designed the Simple Hierarchical Ordered Planner (SHOP) and its successor, SHOP2, with two goals in mind: to investigate research issues in automated planning and to provide some simple, practical planning tools. SHOP and SHOP2 are based on a planning formalism called *hierarchical task network planning*, which is described in more detail in the sidebar, "Automated and HTN Planning," on pages 36–37.

SHOP and SHOP2 are automated planning tools that also serve as an investigative platform for research on automated planning methods.

Most automated planning systems do a trial-and-error search of a large space of possible solutions, and the planner might have to try many different possibilities before finding a plan that works. HTN planners perform this search by applying *HTN methods*, which are essentially forms that describe how to decompose tasks into subtasks. HTN methods generally describe the "standard operating procedures" normally used to perform tasks in some domain; thus, they often correspond well to how users think about problems.

In any trial-and-error search, one of the most important questions is what kind of search-control strategy to use. Unlike most other HTN planners, SHOP and SHOP2 use a search-control strategy called *ordered task decomposition*, which breaks tasks into subtasks and generates the plan's actions in the same order that the plan executor will execute them. So, throughout the planning process, the planner can tell what the state of the world will be at each step of the plan.

To produce the decomposition tree in Figure 1a, for example, SHOP2 would decompose the tasks in the order t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9, t_{10}, and to produce the decomposition tree in Figure 1b, SHOP2 would decompose the tasks in the order t_1, t_2, t_6, t_7, t_3, t_4, t_5, t_8, t_9, t_{10}.

SHOP is similar to SHOP2, but it requires a strict linear ordering on subtasks; hence subtasks cannot be interleaved. For example, SHOP could not produce the decomposition trees in Figure 1 because it wouldn't be able to interleave t_3's and t_6's subtasks. So SHOP2 can describe some planning domains much more easily than SHOP.

By eliminating much uncertainty about the world, ordered task decomposition reduces reasoning complexity. This let us incorporate a great deal of expressive power into the planning system, because it's easier to reason about what *is* true than what might be true. Among other things, SHOP and SHOP2 can do complex inferential reasoning and mixed symbolic/numeric computations, and can call user-supplied subroutines.

In April 2002, SHOP2 achieved high visibility because of its performance in the 2002 International Planning Competition, where it received one of the top four awards. (See http://planning.cis.strath.ac.uk/competition for details.) SHOP2 was one of the three fastest planners in the competition. Its reasoning capabilities let it generate much smaller search spaces than those of most of the other systems. Hence it could solve planning problems—some quite complicated—far more quickly than other systems. SHOP2 solved 899 out of 904 problems, more than any of the other systems.

SHOP and SHOP2 are open source software. Common Lisp and Java implementations are available at www.cs.umd.edu/projects/shop.

Applications of SHOP and SHOP2

SHOP and SHOP2 have been downloaded several thousand times. (As of 11 Dec. 2004, our Web site's log showed 2,163 downloads, but this doesn't include direct downloads from our FTP server rather than through our Web interface. We imagine the total number of downloads exceeds 2,500.) Their significant user base includes government laboratories, industries, and universities, some of which have provided us descriptions of their projects.

Projects in government laboratories

Evacuation planning. The US Naval Research Laboratory's Hierarchical Interactive Case-Based Architecture for Planning (HICAP) is a prototype system for helping experienced human planners develop evacuation plans for people whose lives are in danger. A human expert must supervise evacuation planning: it's unrealistic to expect a planning system to produce good plans by itself, and flawed evacuation plans could yield dire consequences. Therefore, HICAP's top level is a plan editor that lets users edit tasks manually and refine plans interactively.

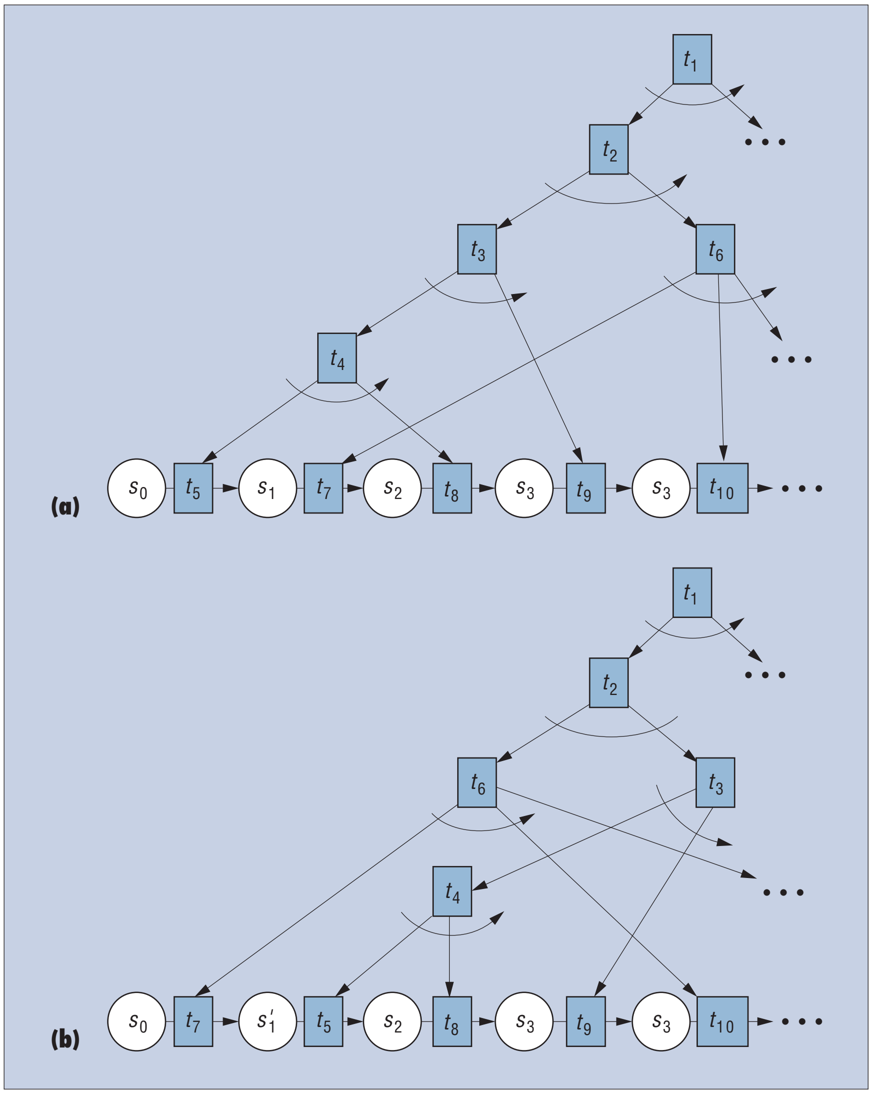
Only part of the knowledge necessary for evacuation planning can be formalized sufficiently for automated planning. In general, the domain description will be incomplete. Standard requirements and operating procedures won't suffice for deriving detailed evacuation plans, which require knowledge based on previous experiences. HICAP's planning module thus includes both generative and case-based planning. SHOP provides the generative component. The case-based component, NaCoDAE, works by retrieving plan fragments from previous evacuations. Within HICAP, NaCoDAE and SHOP are tightly integrated, and each can use the other to decompose tasks into subtasks.

For more information, contact David Aha, Naval Research Laboratory, Washington, DC; www.aic.nrl.navy.mil:80/hicap.

Evaluating terrorist threats. The Naval Research Laboratory's Analogical Hypothesis Elaboration for Activity Detection (AHEAD) project is intended to help intelligence analysts understand and evaluate hypotheses about terrorist threats. The AHEAD system contains models of various kinds of hostile activities, encoded as HTN domain descriptions that the domain author has annotated with additional information about their function. Given a hypothesis, AHEAD uses analogical retrieval to obtain a model of the hostile activity most closely related to the hypothesis. AHEAD invokes SHOP2 using this domain description to produce a plan compatible with the hypothesis. As each planning operator (describing what actions the plan executor can perform) is added to the plan, SHOP2 queries an external evidence database to determine whether the evidence is consistent with that operator. When the evidence is consistent, AHEAD generates an argument in favor of the hypothesis; when the evidence is inconsistent, AHEAD generates a counterargument. The resulting structured argument is presented in a browsable user interface. HTN planning is particularly well suited to this process because HTNs organize behavior into meaningful components at multiple abstraction levels, thus enabling coherent, structured argumentation.

For more information, contact David Aha, Naval Research Laboratory, Washington, DC; www.nrl.navy.mil/aic/iss/ida/projects/ahead/AHEAD.php.

Fighting forest fires. The European Union's COMETS project focuses on developing unmanned aerial vehicle (UAV) control techniques for detecting and monitoring forest fires. As part of this project, researchers at



(a)
t_1
t_2
t_3
t_6
t_4
...
s_0 t_5 s_1 t_7 s_2 t_8 s_3 t_9 s_3 t_{10} ...
(b)
t_1
t_2
t_6
t_3
t_4
...
s_0 t_7 s'_1 t_5 s_2 t_8 s_3 t_9 s_3 t_{10} ...

Figure 1. Two examples of task decomposition with interleaved subtasks. Arrows on the arcs indicate ordering constraints. There is no ordering constraint between t_3 and t_6; hence they may be performed in either order and their subtasks may be interleaved (provided, of course, that each task's preconditions are satisfied).

Automated and HTN Planning

In general, an automated planning system is designed to generate a plan or policy that a plan executor can implement to achieve some set of goals or objectives. Most planning research has focused on *offline* planning, in which the entire plan is formulated before the executor begins implementing it. So at planning time, the planner has no direct information about a plan's success or failure and instead must reason about whether the plan will (or is likely to) succeed or fail.[1]

We can classify automated planning systems into the following categories, based on whether—and how—users can configure them for different planning domains:

- *Domain-independent planners*. The sole input to the planner is a description of a planning problem to solve, and the planning engine must be general enough to work in any planning domain within some large class of planning domains *D*. Historically, *D* was often assumed to be the set of all "classical" planning domains,[1] a class too restricted to include most practical planning applications. However, automated-planning researchers use this assumption less frequently as they become more interested in extending their work beyond classical planning.
- *Tunable domain-independent planners*. These resemble domain-independent planners but can be tuned for better performance in some planning domains. For example, LPG (http://zeus.ing.unibs.it/lpg) can be tuned to run quickly without caring much about the length of the plans it finds, to find short plans even if it takes a long time to find them, or somewhere in between. In the 2002 International Planning Competition, LPG was run with all three settings, producing three result sets.
- *Domain-configurable planners*. The planning engine is domain independent but the input to the planner includes domain-specific *control knowledge*, that is, information to help the planning engine solve problems in the relevant problem domain. These have sometimes been called "hand-tailored" planners. However, because the planning engine is domain independent, the terms "hand tailorable" or "control intensive" are more accurate. Examples of these include hierarchical task network (HTN) planners such as O-Plan,[2] SIPE-2,[3] SHOP, and SHOP2,[4] and planners such as TLPlan[5] and TALplanner[6] whose control knowledge consists of pruning rules.
- *Domain-specific planners*. Tailor-made for a given domain, these planners probably won't work in other domains without major modifications. This class includes most planners that have been deployed in practical applications, for example, the planning algorithm for declarer play used in *Bridge Baron*.[7]

HTN planning is a well-known approach for domain-specific and domain-configurable planning. In HTN, the planning system formulates a plan by decomposing tasks (symbolic representations of activities to be performed) into smaller and smaller subtasks until it reaches primitive tasks that the plan executor can perform directly. The basic idea emerged in the mid-1970s.[1]

```
method travel-by-foot
    precond:   $distance(x,y) \le 2$
    task:      travel($a, x, y$)
    subtasks:  walk($a, x, y$)

method travel-by-taxi
    task:      travel($a, x, y$)
    precond:   $cash(a) \ge 1.5 + 0.5 \times distance(x, y)$
    subtasks:  call-taxi($a, x$) $\rightarrow$ ride($a, x, y$) $\rightarrow$ pay-driver($a, x, y$)

operator walk ($a, x, y$)
    precond:   $location(a) = x$
    effects:   $location(a) \leftarrow y$

operator call-taxi($a, x$)
    effects:   $location(taxi) \leftarrow x$

operator ride-taxi($a, x$)
    precond:   $location(taxi) = x, location(a) = x$
    effects:   $location(taxi) \leftarrow y, location(a) \leftarrow y$

operator pay-driver($a, x, y$)
    precond:   $cash(a) \ge 1.5 + 0.5 \times distance(x, y)$
    effects:   $cash(a) \leftarrow cash(a) - 1.5 + 0.5 \times distance(x, y)$
```

Figure A. Pseudocode representation of a simple travel-planning domain. Left-arrows denote assignments of values to state variables; right-arrows are ordering constraints.

An HTN planning problem consists of the initial state (a symbolic representation of the state of the world at the time the plan executor will begin executing its plan), the initial task network (a set of tasks to be performed, along with some constraints that must be satisfied), and a domain description that contains the following:

- A set of *planning operators* describing what actions (that is, what primitive task types) the plan executor can perform. Each operator can have a set of preconditions that must be true before it can be executed and a set of effects that will occur afterward. An action (or synonymously, a primitive task) is an operator instance, produced by assigning values to an operator's parameters.
- A set of *methods* describing possible ways of decomposing tasks into subtasks. These are the "standard operating procedures" normally used to perform tasks in the domain. Each method can have a set of constraints that the world state must satisfy before the planner can apply the method.
- Optional information such as definitions of auxiliary functions and of axioms for inferring conditions not explicitly

LAAS, a government research laboratory in Toulouse, France, are developing a distributed architecture in which each UAV will contain a generic "decisional node" consisting of a supervisor and a planner.

Within each decisional node, SHOP2 acts as the symbolic planner. Its forward-chaining capability integrates the planning activity with specialized software for estimating basic UAV operations' costs, time, and so on. To perform temporal reasoning, the LAAS researchers use the same time-stamping technique we developed for temporal planning with SHOP2 in the 2002 International Planning Competition.[1] The researchers expect to have simulation results soon and then to run experiments using Karma, LAAS's robotic blimp.

mentioned in states of the world.

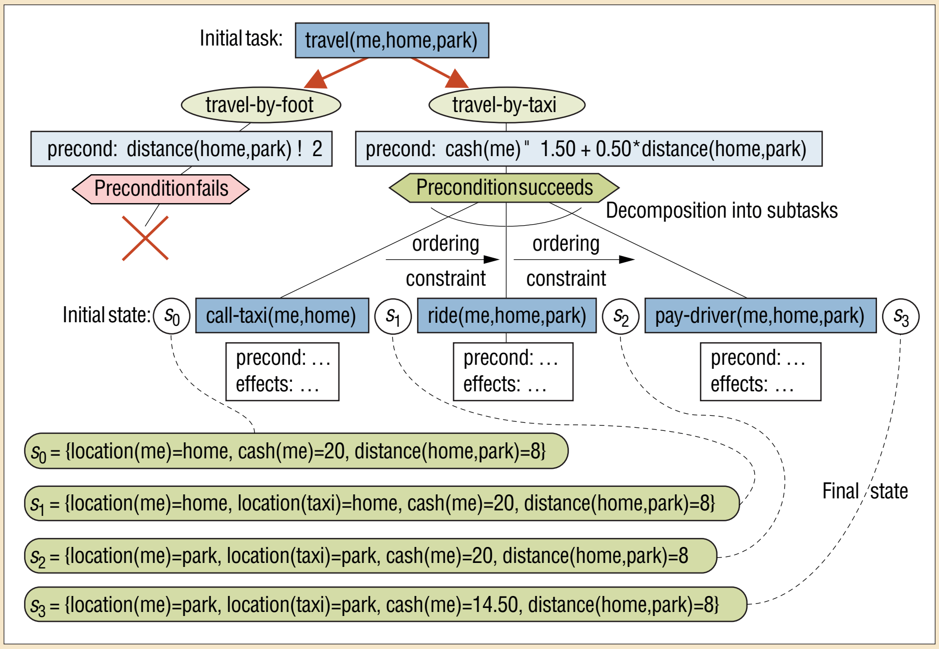
For each nonprimitive task, the planner chooses an applicable method and instantiates it to decompose the task into subtasks. For each primitive task, the planner chooses an applicable operator and instantiates it to produce an action. If all constraints are satisfied, the planner has found a solution plan; otherwise, it will need to backtrack and try other methods or instantiations.

For example, Figure A gives a pseudocode representation of a simple planning domain that presents two ways to travel from one location to another: by foot and by taxi. The travel-by-foot method has one constraint: the distance from the starting point to the destination must be less than or equal to two miles. If the method is applicable, it decomposes the task into a single subtask: walk to the park. The travel-by-taxi method has one constraint which is also a precondition: the traveler must have enough cash to pay the taxi driver. If the method is applicable, it decomposes the task into three subtasks: call a taxi, ride to the park, and pay the driver. All of the subtasks are primitive; that is, the traveler is expected to know how to accomplish them directly.

Now, suppose that in the initial state, you're at home, you have $20, and you want to travel to a park eight miles away. To plan how to travel to the park, you first try the travel-by-foot method, but it doesn't apply because the park is more than two miles away. Next, you try the travel-by-taxi method. Its precondition is satisfied, so the method produces a sequence of three subtasks, with a constraint saying they are to be performed in the following order:

1. call a taxi to your home,
2. ride in it to the park, and
3. pay the driver $5.50.

The subtasks all are primitive, that is, each corresponds to an action. The first action has no preconditions, so it is applicable and produces a state s_1 identical to the initial state except that *location*(*taxi*) = *home*. This state satisfies the preconditions of the second action, which in turn satisfies that of the third action, so you have a solution plan. Figure B shows the final state after this plan executes.



Initial task:
travel(me,home,park)
travel-by-foot
travel-by-taxi
precond: distance(home,park) ! 2
precond: cash(me)" 1.50 + 0.50*distance(home,park)
Preconditionfails
Preconditionsucceeds
Decomposition into subtasks
ordering constraint
ordering constraint
Initial state: s0
call-taxi(me,home)
s1
ride(me,home,park)
s2
pay-driver(me,home,park)
s3
precond: … effects: …
precond: … effects: …
precond: … effects: …
s0 = {location(me)=home, cash(me)=20, distance(home,park)=8}
s1 = {location(me)=home, location(taxi)=home, cash(me)=20, distance(home,park)=8}
s2 = {location(me)=park, location(taxi)=park, cash(me)=20, distance(home,park)=8
s3 = {location(me)=park, location(taxi)=park, cash(me)=14.50, distance(home,park)=8}
Final state

Figure B. Solving a planning problem in the travel-planning domain.

References

1. M. Ghallab, D. Nau, and P. Traverso, *Automated Planning: Theory and Practice*, Morgan Kaufmann, 2004.
2. A. Tate, B. Drabble, and R. Kirby, *O-Plan2: An Architecture for Command, Planning and Control*, Morgan Kaufmann, 1994.
3. D.E. Wilkins, *Practical Planning: Extending the Classical AI Planning Paradigm*, Morgan Kaufmann, 1988.
4. D. Nau et al., "SHOP2: An HTN Planning System," *J. Artificial Intelligence Research*, Dec. 2003, pp. 379–404.
5. F. Bacchus and F. Kabanza, "Using Temporal Logics to Express Search Control Knowledge for Planning," *Artificial Intelligence*, vol. 116, nos. 1–2, 2000, pp. 123–191.
6. J. Kvarnström and P. Doherty, "TALplanner: A Temporal Logic Based Forward Chaining Planner," *Annals of Mathematics and Artificial Intelligence*, vol. 30, 2001, pp. 119–169.
7. S.J.J. Smith, D.S. Nau, and T. Throop, "Computer Bridge: A Big Win for AI Planning," *AI Magazine*, vol. 19, no. 2, 1998, pp. 93–105.

For more information, contact Jeremi Gancet, Simon Lacroix, and Raja Chatila, LAAS/CNRS, Toulouse, France; www.comets-uavs.org.

Software systems integration. The US National Institute of Standards and Technology (NIST) is using SHOP2 in a project for automating software systems integration tasks. So far, the particular example they've used is based on General Motors' ebXML-based rental-car-buying interfaces. These interfaces let buyers search for and purchase cars of a particular make, model, and year. However, the interfaces make it difficult for a buyer to get a summary of various cars at various locations (for example, to minimize costs). NIST's code reads the seller's ebXML BPSS (business process specification schema) and produces a planning problem in which SHOP determines the transaction sequence against the seller's interfaces to achieve the buyer's objective.

For more information contact, Peter Denno, NIST, Gaithersburg, Md; www.mel.nist.gov/proj/mee.htm.

Industry projects

Controlling multiple UAVs. Smart Information Flow Technologies (SIFT) is using a modified version of the SHOP2 planner in a UAV control system in their Playbook-Enhanced Variable Autonomy Control System (PVACS) project, with funding from DARPA through a small-business innovation research contract. SIFT's Playbook control system allows time-pressured users, who are not UAV operators, to request reconnaissance missions using high-level tasking commands, modeled on how people delegate tasks to human subordinates. The SIFT Playbook supports interactions through both PDA and desktop or laptop interfaces. It translates brief, general user commands into specific control actions suitable for execution. The Playbook's Executive provides high-level closed-loop monitoring and implementation of the Playbook's plans, controlling multiple UAVs through the Variable Autonomy Control System (VACS) Ground Control Station (GCS), developed by Geneva Aerospace.The Playbook currently operates these UAVs in a high-fidelity simulation environment, but it uses the same interface to control the simulated UAVs through the VACS GCS as that used to direct VACS UAVs in real flight operations.

The modified SHOP2 planner plays a key role in SIFT's Playbook, translating the user's high-level task specifications into a sequence of commands that UAVs can execute. SIFT's plan library contains tasks for multiple reconnaissance missions, for both rotorcraft and fixed-wing UAVs. Robert Goldman at SIFT has developed an augmented version of SHOP2 that generates temporal plans, including durative actions, and provides more knowledge-engineering and debugging support.

For more information, contact Robert Goldman, SIFT, Minneapolis, Minn; www.sift.info/English/projects/PVACS.htm.

Evaluation of enemy threats. Lockheed Martin Advanced Technology Laboratories, in collaboration with the Army Research Laboratory, is using SHOP in a project that attempts to evaluate possible enemy threats. The project uses SHOP to decompose higher-level tasks such as "attack blue convoy" into sequences of operations such as move red-tank1 to location2, …, fire red-tank1 at blue-convoy. Project details are confidential.

For more information, contact Benjamin Grooters and Sergio Gigli, Lockheed Martin ATL, Cherry Hill, N.J.

Location-based services. Sony Electronics has used SHOP in a project aimed at developing mobile geographical information system (GIS) devices to help people plan errands that take them to different locations. Project details are confidential.

For more information, contact Mark Plutowski, Sony Electronics Inc., San Jose, Calif.

The SHOP/CCBR system was developed to investigate using HTN planning techniques. The system is a straightforward SHOP extension that uses cases to decompose tasks.

Material selection for manufacturing. Infocraft is developing a system that uses SHOP2 for materials selection in a continuous manufacturing process to produce activated carbon from charcoal. The carbon's desired properties (specifically, its grade size and adsorption level) will vary from one run to another, as will characteristics of different charcoal supplies. The project uses SHOP2 to select which charcoal supplies will most reliably produce activated carbon with a desired set of properties. SHOP2's numerical and axiomatic reasoning abilities are essential for this project: adsorption levels are represented as real numbers, and grade sizes are represented as normal distributions.

For more information, contact Nuwan Waidyanatha, Infocraft, Sri Lanka; www.infocraft.lk.

University projects

Automated composition of Web services. A Web service is a Web-accessible chunk of functionality with an interface described in a machine-readable standard format. Web services are designed to be composed in loosely coupled workflows of varying complexity to provide functionality that none of the component services could provide alone.

The OWL-S (Web Ontology Language for Services, formerly called DAML-S) language for semantic markup of Web services describes services as complex or atomic processes with preconditions and effects. This permits translating the OWL-S process-model constructs directly to SHOP2 methods and operators, and University of Maryland researchers have developed an algorithm to do so. Users apply SHOP2 to solve service composition problems by telling it to find a plan for an HTN task that corresponds to the composite process.

For more information, contact Evren Sirin and James Hendler, Dept. of Computer Science, University of Maryland; www.mindswap.org/~evren/composer.

Project planning. The SHOP/CCBR system was developed at Lehigh University to investigate using HTN planning techniques to support project management. The system is a straightforward SHOP extension that uses *cases* to decompose tasks. Cases are similar in structure to methods, the main difference being that cases include preference information for ranking applicable cases. SHOP/CCBR uses a communication module to interact with Microsoft Project, a commercial project management tool that can display the HTN decompositions generated with SHOP's hierarchical planning algorithm. The ongoing work involves developing algorithms to capture cases automatically from user interactions with Microsoft Project.

For more information, contact Héctor Muñoz-Avila, Computer Science and Engineering, Lehigh University; www.cse.lehigh.edu/~munoz/projects/KBPP.

Statistical goal recognition in agent systems. For an agent to perform effectively in a multiagent environment, it must be able to infer other agents' goals. University of Rochester researchers are developing a statistical approach to goal recognition using machine-learning techniques. The learning requires a labeled "plan corpus" of plans and their associated goals. To generate such plan corpora stochastically, the researchers are using a modified version of SHOP2 that makes random choices at every point where more than one possible decision is available.

For more information, contact Nate Blaylock and James Allen, Computer Science Dept., University of Rochester; www.cs.rochester.edu/research/cisd/projects/goalrec.

Distributed planning. Researchers at the University of Sherbrooke are developing a general approach for turning backtracking-search-based planners into distributed planning systems that run on cluster networks. The basic idea is to distribute backtrack search points to different network processes. To implement their approach, the researchers have developed DSHOP, a distributed version of SHOP.

For more information, contact Froduald Kabanza, Département d'informatique Université de Sherbrooke, Canada; www.planiart.usherbrooke.ca/kabanza.

Additional university projects. Worldwide, SHOP and SHOP2 have been used in many more college and university projects than we can mention. Here are some notes about a few of them.

- Drexel University regularly uses SHOP and SHOP2 in their introductory AI class to teach planning, and in their Knowledge-Based Agents course to do agent reasoning, service composition, and the like. Contact William Regli, regli@drexel.edu.
- The National University of Colombia in Medellin, Colombia, is developing a system that uses SHOP2 to automatically create virtual courses from existing educational material. Contact Néstor Daro Duque Mendez, nduque@nevado.manizales.unal.edu.co.
- The Technical University of Cluj-Napoca in Romania has used SHOP for an e-commerce application that builds bidding plans in a modified version of the Trading Agent Competition. Contact Adrian Groza, adrian.groza@email.ro.
- Trinity College Dublin is using SHOP2 in a Web-service composition project somewhat similar to ours. Contact Dónal Murtag, domurtag@cs.tcd.ie.
- The University of Maryland's Aerospace Engineering Department incorporates SHOP2 as the planning component in an architecture that combines task planning, real-time scheduling, and motion/trajectory planning. Contact Ella Atkins, atkins@glue.umd.edu.
- Villanova University has implemented SHOP2 in a mock spacecraft-mission scenario to study how the planning system can use the density, distribution, and overall layout of environmental obstacles to compute and predict the best optimization technique. Contact Filip Jagodzinski, filip.jagodzinski@villanova.edu.
- University of Arizona is using SHOP to generate process alternatives within a case-based reasoning framework for business workflow management. Contact Madhu Therani, madhu@email.arizona.edu.

Future work

We've been pleasantly surprised at the extent to which people have begun using SHOP and SHOP2 in their research and development projects. Their availability as open source software makes it easy for users to find and fix bugs and adapt the tools for their own purposes, but we believe their popularity also reflects their use of planning methods that complement how people think about generating plans. We have many ideas for extensions and improvements. (We present relevant citations elsewhere.[2])

We've been pleasantly surprised at the extent to which people have begun using SHOP and SHOP2. Their availability as open source software makes it easy for users to adapt the tools.

Automated learning of planning domains

A great challenge in using any planning system to solve real-world problems is acquiring the domain knowledge the system will need to plan effectively. We can divide this domain knowledge into two types:

- *Domain definition,* that is, domain states and operators. Researchers usually assume such knowledge will be provided a priori, but some real-world domains likely have only a partial or approximate description of the planning domain. For example, it's easy to say that a drilling operation will drill a hole but much harder to say how far the hole will deviate from perfect straightness and roundness or to model the conditions under which the drill bit is likely to break.
- *Control knowledge*, or information to guide the planning process. In HTN planners such as SHOP and SHOP2, this knowledge consists of methods. In planning systems such as TLPlan and TALplanner (see the sidebar), it consists of rules for pruning the search space.

We're working on ways to acquire HTN methods automatically by having the planning system learn them from plan traces. We have developed a general formal framework for learning HTN methods and a supervised learning algorithm, CAMEL, based on this formalism. Theoretical and experimental studies of its soundness, completeness, and convergence properties suggest that CAMEL could prove useful in real-world applications.[3]

Compiling planning domains

We can view a domain-configurable planner as an *interpreter* of its domain description language: given a domain description D and a planning problem P, the planner invokes the methods and operators of D interpretively on P.

An alternative approach is to write a *compiler* for the domain description language: the input is a domain description D, and the output is a domain-specific planning program for D that can be run directly on any planning problem P in D. The advantages of this approach are analogous to compilation's advantages over interpretation in conventional programming languages. Compiling domain descriptions directly into low-level executable code lets us do implementation-level optimizations that are otherwise impossible, and unexplored in previous AI planning research. We can couple these optimizations with other speed-up techniques (such as domain analysis and other automated domain information synthesis methods) to obtain additional speedups.

JSHOP2, a Java implementation of SHOP2 that we're now developing, uses this domain-compilation technique. Our preliminary experimental results suggest that the compilation technique increases the planner's efficiency by an order of magnitude. A technical report on this topic is available at www.cs.umd.edu/~okhtay/jshop2doc.pdf, and JSHOP2 itself is available at www.cs.umd.edu/projects/shop.

Planning under uncertainty

Automated-planning research has traditionally assumed that each action is *deterministic*—that only one possible outcome exists. But it is often more realistic to let actions be *nondeterministic*, having multiple possible outcomes. For example, an action's outcome might vary because of other agents' responses or random environmental changes.

We've developed a general technique for adapting planners such as SHOP, SHOP2, TLPlan, and TALplanner to deal with nondeterministic actions. We have shown both theoretically and experimentally[4] that our approach can produce exponential speedups over previous algorithms for planning in nondeterministic environments.

We're also extending our approach to work in Markov decision processes, in which the actions have probabilistic outcomes. Our experiments show we can obtain exponential speedups here, too.

Planning with distributed information sources

Planning researchers typically assume the planning system is *isolated*: it begins with a complete description of the planning problem and requires no interaction with the external world during the planning process. In many practical situations, such an assumption is clearly unrealistic: the planner may need information from external sources during planning.

We have developed a formalism for *wrappers* to place around conventional (isolated) planners to replace some of the planner's memory accesses with queries to external information sources. When appropriate, the wrapper can automatically backtrack the planner to a previous point in its operation. We mathematically and experimentally analyzed several different query-management strategies for these wrappers, such as when to issue queries, and when and how to backtrack the planner.[5] Results suggest that domain-configurable planners such as SHOP2 are likely better suited than other planners for planning with volatile information.

Performance improves even more if a planner can make *nonblocking* queries to external information sources—that is, continue exploring other parts of its search space while waiting for the query's response. We have described a modified version of SHOP2 that works in this way and shown experimentally that this dramatically improves the planner's performance.[6]

One weakness of SHOP2 is that its methods and operators do not explicitly represent time and concurrency—two things that would be essential, for example, to permit concurrent actions of different agents (such as multiple subordinates or team members). It might be possible in some cases to overcome this limitation by writing planning operators that explicitly assert timestamp information into the current state of the world. On the other hand, it clearly would be advantageous to have a more comprehensive way to represent and reason about time and concurrency. We hope to address this problem in our future work.

Meanwhile, both SHOP and SHOP2 are helping users apply computers to solve complex planning problems, by providing domain-independent planning engines that can make use of user-provided knowledge about specific planning domains. ■

Acknowledgments

We are grateful to our user community for the information they gave us about how they use SHOP and SHOP2 in their projects. This work was supported in part by the following grants and contracts: US Air Force Research Laboratory F30602-00-2-0505, Army Research Laboratory DAAL 0197K0135, Naval Research Laboratory N00173021G005, and National Science Foundation IIS0412812. The opinions expressed in this paper are the authors' and do not necessarily reflect those of the funders.

References

1. D. Nau et al., "SHOP2: An HTN Planning System," *J. Artificial Intelligence Research*, vol. 20 Dec. 2003, pp. 379–404.

2. D. Nau et al., *Applications of SHOP and SHOP2*, tech. report CS-TR-4604, UMIACS-TR-2004-46, Univ. of Maryland, 2004; www.cs.umd.edu/~nau/papers/nau04applications.pdf.

3. O. Ilghami et al., "CaMeL: Learning Method Preconditions for HTN Planning," *Proc. 6th Int'l Conf. AI Planning and Scheduling* (AIPS 02), AAAI, 2002, pp. 131–142.

4. U. Kuter and D. Nau, "Forward-Chaining Planning in Nondeterministic Domains," *Proc. 19th Nat'l Conf. Artificial Intelligence* (AAAI 04), AAAI Press/MIT Press, 2004, pp. 513–518.

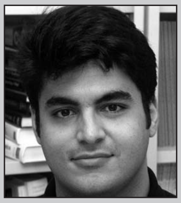
5. T.-C. Au, D. Nau, and V. Subrahmanian, "Utilizing Volatile External Information During Planning," *Proc. 16th European Conf. Artificial Intelligence* (ECAI 04), IOS Press, 2004, pp. 647–651.

6. U. Kuter et al., "Information Gathering During Planning for Web Services Composition," *Proc. 3rd Int'l Semantic Web Conf.* (ISWC 04), LNCS 3298, Springer-Verlag, 2004, pp. 335–349.

The Authors

Dana Nau is a professor of both computer science and systems research at the University of Maryland. His research interests include AI planning and searching, and computer-integrated design and manufacturing. He received his PhD in computer science from Duke University. He is an AAAI Fellow and recently coauthored *Automated Planning: Theory and Practice*, the first comprehensive textbook on automated planning. Contact him at the Dept. of Computer Science, A.V. Williams Building, University of Maryland, College Park, MD 20742; nau@cs.umd.edu.

Tsz-Chiu Au is a doctoral student of the Department of Computer Science at the University of Maryland. His research interests include AI planning, problem solving by searching, and constraint satisfaction. He received his BEng in computer science from the Hong Kong University of Science and Technology. Contact him at the Dept. of Computer Science, A.V. Williams Building, University of Maryland, College Park, MD 20742; chiu@cs.umd.edu.

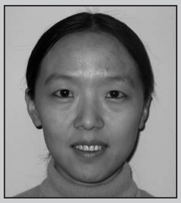


Okhtay Ilghami is a doctoral student and research assistant at the University of Maryland. His research interests include AI planning and machine learning. He received his MSc in computer engineering from the University of Maryland at College Park. Contact him at the Dept. of Computer Science, A.V. Williams Building, University of Maryland, College Park, MD 20742; okhtay@cs.umd.edu.

Ugur Kuter is a doctoral candidate in the Department of Computer Science at the University of Maryland. His current research interests include planning and decision making under conditions of uncertainty. Kuter received his MSc in computer engineering from Middle East Technical University, Ankara, Turkey. He is a student member of AAAI. Contact him at Dept. of Computer Science, A.V. Williams Building, University of Maryland, College Park, MD 20742; ukuter@cs.umd.edu.

Héctor Muñoz-Avila is an assistant professor in the Department of Computer Science and Engineering at Lehigh University. His research interests include case-based reasoning and planning. He received his PhD from the University of Kaiserslautern. Contact him at the Dept. of Computer Science and Engineering, 19 Memorial Drive West, Leigh University, Bethlehem, PA 18015; munoz@cse.lehigh.edu.

J. William Murdock is a member of the Knowledge Structures Group in IBM's Watson Research Center. His research area is knowledge-based AI, and his interests include reflection, planning, explanation, adaptation, learning, and natural-language understanding. He received his PhD in computer science from the Georgia Institute of Technology. He is a member of AAAI and the ACM. Contact him at IBM Watson Research Center, 19 Skyline Dr., Hawthorne, NY 10532; murdockj@us.ibm.com.



Dan Wu is a product support manager at MicroStrategy. Her research interests include AI planning and its application to Semantic Web services. She received an MS in computer science from the University of Maryland at College Park. Contact her at MicroStrategy, 1861 International Dr., McLean, VA 22102; dwu@microstrategy.com.

Fusun Yaman is a doctoral candidate in the Department of Computer Science at the University of Maryland. Her research interests are temporal planning, plan management, and motion theory. She received an MS in computer science from the University of Maryland. She is a member of AAAI. Contact him at the Dept. of Computer Science, A.V. Williams Building, University of Maryland, College Park, MD 20742; fusun@cs.umd.edu.