In *AAAI-98/IAAI-98 Proceedings*, pages 1127-1133, 1998.

Control Strategies in HTN Planning: Theory Versus Practice

Dana S. Nau **Stephen J. J. Smith** **Kutluhan Erol**

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

Department of Mathematics and Computer Science, Hood College, Frederick, MD, USA, sjsmith@nimue.hood.edu

Intelligent Automation Inc., 2 Research Place #202, Rockville, MD 20850, kutluhan@i-a-i.com

Abstract

AI planning techniques are beginning to find use in a number of practical planning domains. However, the backward-chaining and partial-order-planning control strategies traditionally used in AI planning systems are not necessarily the best ones to use for practical planning problems. In this paper, we discuss some of the difficulties that can result from the use of backward chaining and partial-order planning, and we describe how these difficulties can be overcome by adapting Hierarchical Task-Network (HTN) planning to use a total-order control strategy that generates the steps of a plan in the same order that those steps will be executed. We also examine how introducing the total-order restriction into HTN planning affects its expressive power, and propose a way to relax the total-order restriction to increase its expressive power and range of applicability.

Introduction

Although there has been much recent interest in exploring alternative control strategies for AI planning systems, one aspect of the control strategy that is usually taken for granted is the use of partial-order planning and backward chaining. However, a backward-chaining partial-order-planning strategy is not necessarily the best kind of strategy to use in practical planning problems. Two examples come from our recent use of HTN planning in two very different application domains:

- As reported in *The New York Times* and *The Washington Post*, a new version of Great Game Products' *Bridge Baron* program won the 1997 *Baron Barclay World Bridge Computer Challenge*. The competition, which was hosted by the American Contract Bridge League (ACBL) in July 1997, is effectively the world championship compeition for computer bridge programs. The first two authors helped develop this new version of the Bridge Baron (Smith *et al.*, 1996), incorporating HTN planning techniques into it to improve its declarer play.
- EDAPS (Hebbar *et al.*, 1996) is an integrated system for designing and planning the manufacture of microwave transmit/receive modules. EDAPS uses HTN planning to generate process plans for proposed designs, and evaluates these plans to provide feedback to the designer about the manufacturability of the design. The first author is currently working with Northrop Grumman to develop EDAPS into a product for use in their manufacturing facility in Baltimore.

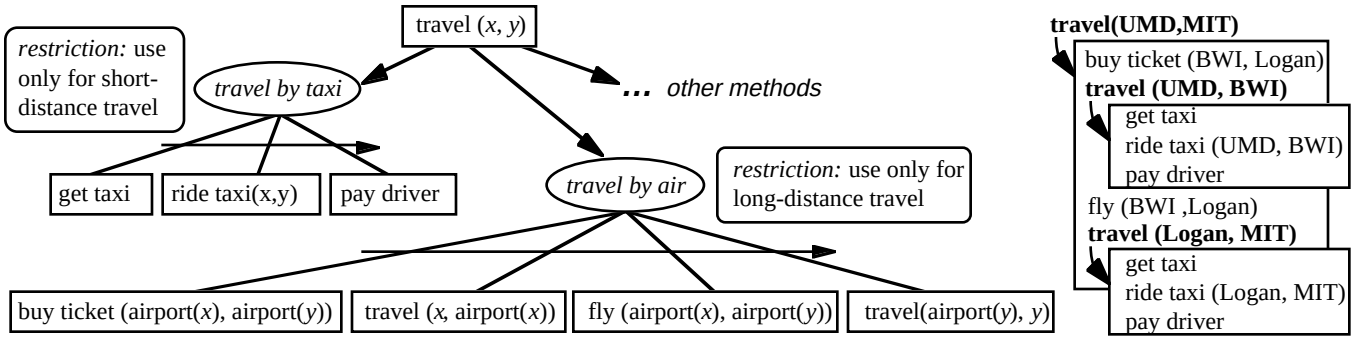
These application domains were not amenable to the partial-order planning and backward chaining techniques used in most AI planning systems. Instead, we developed a total-order planning control strategy that expands tasks in the same order that they will be executed. This control strategy worked quite well in both application domains.

This experience has led us to question how well the reasons normally given for using partial-order planning and backward chaining apply to real-world planning domains—and in this paper, we try to answer that question. In particular, we do the following:

1. We examine some of the assumptions behind the use of backward chaining and partial-order planning, and conclude that these assumptions do not always apply to HTN planning in real-world domains.
2. We describe some of the difficulties that can be caused by backward chaining and partial-order planning, and explain how total-order HTN planning can overcome those difficulties.
3. While total-order HTN planning is strictly more expressive than planning with STRIPS-style operators, it is also strictly less expressive than unrestricted HTN planning. We propose a way to relax the "total-order" restriction, to produce a sound and complete forward-search approach to HTN planning that has the same expressive power as unrestricted HTN planning.

Overview of HTN Planning

The basic ideas used in HTN planning were originally developed more than 20 years ago (Sacerdoti 1977; Tate 1977). To create plans, HTN planning uses *task decomposition*, in which the planning system decomposes tasks into smaller and smaller subtasks until primitive tasks are found that can be performed directly. HTN planning systems have knowledge bases containing *methods*. As shown in Figure 1(a), each method includes a prescription for how to decompose some task into a set of subtasks,



(a) methods (b) plan

Figure 1: Methods for traveling, and their use in developing a plan for traveling from U. of Maryland to MIT.

with various restrictions that must be satisfied in order for the method to be applicable, and various constraints on the subtasks and the relationships among them. Given a task to accomplish, the planner chooses an applicable method, instantiates it to decompose the task into subtasks, and then chooses and instantiates other methods to decompose the subtasks even further, as illustrated in Figure 1(b). If the constraints on the subtasks or the interactions among them prevent the plan from being feasible, the planning system will backtrack and try other methods.

Formal analyses of HTN planning have shown that it is strictly more expressive than planning with STRIPS-style operators (Erol *et al*. 1994b), and have established properties such as soundness and completeness (Erol *et al.* 1994a), complexity (Erol *et al*. 1996), and the efficiency of various control strategies (Tsuneto *et al*. 1996, 1997). A domain-independent HTN planner is available at <http://www.cs.umd.edu/projects/plus/umcp/manual> for experimental use, and domain-specific HTN planners are being developed for several practical applications (Aarup *et at*. 1994; Hebbar *et al.* 1996; Smith *et al.* 1997; Smith *et al.* 1998; Wilkins & Desimone 1994).

Two Applications of HTN Planning

Contract Bridge

Although computer programs have done very well in games such as chess, checkers, and Othello (Schaeffer 1993; Korf 1994), human experts still outperform computers in the game of bridge. One reason for this is the difficulty of adapting traditional game-tree search techniques to imperfect-information games. Bridge players normally don't know what cards are in the other players' hands (except for what cards are in the dummy's hand)—and thus each player has only partial knowledge of the state of the world, the possible actions, and their effects. A game tree that represents all sequences of actions that *might* be possible would be much too large to be searched within the time available during a bridge game.

For declarer play in bridge, we have developed a different approach, that grows out of the observation that bridge is a game of planning. The bridge literature describes a number of tactical and strategic schemes (such as finessing, ruffing, and crossruffing) that people combine into plans for how to play their bridge hands. We have taken advantage of the planning nature of bridge, by adapting and extending some ideas from HTN planning.

To represent the tactical and strategic schemes of card-playing in bridge, we use structures similar to HTN methods—but modified to represent multi-agency and uncertainty. Each multi-agent method encodes a portion of some bridge tactic (such as finessing, ruffing, cross-ruffing, cashing out, etc.). To generate game trees, we apply HTN decomposition to these multi-agent methods. This produces a game tree in which each branch represents a move that fits into some coherent strategy. In comparison with a brute-force game-tree search, this gives us a smaller branching factor and a much smaller search tree: our approach generates game trees containing only about 305,000 nodes in the worst case and 26,000 nodes on the average, as compared to 5.55×10^{44} nodes in the worst case and 10^{24} nodes on the average if we had generated a conventional game tree. Thus, our algorithm can search the game tree all the way to the end, to predict the likely results of the various sequences of cards it might play.

We have incorporated our HTN planning techniques for declarer play into the latest version of Great Game Products' Bridge Baron program, *Bridge Baron 8*. As reported in *The New York Times* (Truscott 1997) and *The Washington Post* (Chandrasekaran 1997), a pre-release version of Bridge Baron 8 won the latest world-championship competition for computer bridge programs, the 1997 *Baron Barclay World Bridge Computer Challenge*. Table 1 shows how each of the contenders placed in the competition.

The official version of Bridge Baron 8 went on the

Table 1: The contenders in the *Baron Barclay World Bridge Computer Challenge*, and their final places.

Program	Country	Performance
Bridge Baron	USA	1st place
Q-Plus	Germany	2nd place
MicroBridge 8	Japan	3rd place
Meadowlark	USA	4th place
GIB	USA	5th place

market in October 1997. During the last three months of 1997, it was purchased by more than 1000 people. More information about Bridge Baron 8 appears elsewhere in this conference proceedings (Smith *et al* 1998).

Process Planning for Microwave Modules

Microwave transmit/receive modules are complex electronic devices that are used in such applications as radars and satellite communications. These modules include a number of analog and digital components mounted on a substrate. Since these modules operate in the 1-20 GHz range, the location and orientation of the components on the substrate—and the dimensions and location of the printed wiring—affect the electrical performance. Thus the design of these modules is a complex task.

As with many manufactured products, the cost of manufacturing microwave modules is largely determined by decisions made while the modules are being designed. Thus, while a designer is producing a design, it is important to consider how the design decisions will affect not only the design's performance but also its potential manufacturing cost.

EDAPS (Electro-Mechanical Design and Planning System) is a toolkit to aid in this task. It integrates electronic design with mechanical design by incorporating a commercial electronic CAD tool (Hewlett Packard's *EEsof*) and a mechanical CAD tool (Bentley Systems' *Microstation*). Furthermore, it uses an HTN planning system to generate manufacturing process plans from the CAD models. Manufacturing a microwave module involves both mechanical operations (to produce holes and pockets in the substrate) and electronic operations (such as artwork generation and soldering operations). Each of these is discussed briefly below:

- For complex machined parts it can be quite difficult to generate good process plans (Nau *et al*. 1995). In particular, if the part geometry is complicated, it can be very difficult to determine what machining operations to use in what order, and how best to clamp the part to hold it in place during machining. However, for microwave modules the geometry of the substrate is relatively simple, and thus the task is considerably easier. Since there are few interactions among the features to be machined on the substrate, it is possible to find a one-to-one correspondence between machined features in the substrate and the machining operations that will be used to make them. It is easy to encode the machining processes using HTNs, by representing each process with an HTN method, and easy to instantiate these methods to find process orderings that work.
- For electrical operations such as artwork generation and soldering, the process structure is more complicated than for the mechanical operations—but it decomposes naturally into tasks and subtasks that are performed in a fixed sequence. This decomposition hierarchy can be represented quite naturally using HTNs, making it easy to develop plan details depending on the details of the design.

Once EDAPS has generated a process plan, it does calculations to estimate the plan's time, cost, and product quality. By generating and comparing several plans and displaying the pareto-optimal ones to the user, this allows the user to select the best plan and to make an informed decision about the manufacturability of the design.

Comparison with Conventional Planning

Both the Bridge Baron and EDAPS required some extensions to the usual formulation of HTN planning:

- The Bridge Baron needed ways to represent and reason about possible actions by other agents (such as the opponents in a bridge game), as well as uncertainty about their capabilities (for example, lack of knowledge about what cards they have).
- EDAPS needed a way for the planner to interact with CAD modelers to get information about the product to be manufactured.

In order to incorporate these extensions, we found it necessary to restrict the ways in which the Bridge Baron and EDAPS go about generating plans. Both the Bridge Baron and EDAPS use total-order planning, and both expand tasks in a "left-to-right" order, i.e., in the same order that the tasks will be executed later. For example, in Figure 1, this control strategy would plan the steps in exactly the order in which they appear in Figure 1(b).

In contrast, most planning systems described in the AI planning literature operate using partial-order planning and backward chaining. Below, we examine the assumptions that have led to this preference, and explain why those assumptions do not apply to the Bridge Baron and EDAPS.

Backward Chaining

Most AI planning systems use some form of goal-directed search to improve the efficiency of planning by reducing the branching factor of the search space. One of the most common kinds of goal-directed search is backward chaining, in which the planner chooses some condition that needs to be achieved (either because it is specified as a goal or is a precondition for an operator that is already in the plan) but has not yet been achieved, and looks for a way to achieve it.

For example, in the blocks-world problem shown in

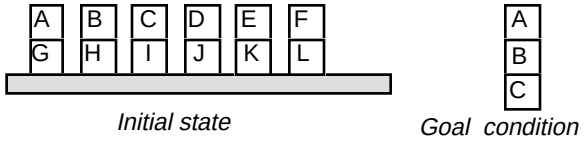


Figure 2. A planning problem that illustrates the motivation for backward chaining.

Figure 2, suppose the basic available planning operator is the operator *move(X,Y)*, which can move *X* to *Y* if *X* is a clear block and *Y* is either the table or a clear block other than *X*. Then in the initial state there are 36 applicable instances of the "move" operation, but only one of them (namely *move(B,C)*) is of any use in achieving the goal. If a planner were to develop plans by searching forward from the initial state and considering all operator instances that are applicable at each point, it might easily search down lots of wrong paths before finding the right one. Instead, the planner focuses on achieving the goal conditions *on(A,B)* and *on(B,C)*; thus, it will immediately look at the only planning operations that can possibly be relevant: *move(A,B)* and *move(B,C)*. Some of the assumptions embodied in this example include the following:

- The number of operators that are applicable at each state of the world is generally higher than the number of operators capable of producing that state of the world. Thus, if we do forward chaining, the search space will have a higher branching factor than if we do backward chaining.
- Many of the paths in the search space can be much longer than the length of the plan we are trying to find. If we go down the wrong path, it may take a long time to discover this and backtrack to another path, so it is very important to minimize the number of paths we explore.

In contrast, the domains addressed by the Bridge Baron and EDAPS have the following characteristics:

- There may be many paths to goal states, and we want to find the optimal one. Most of the paths in the search space have length similar to the length of the plan we are trying to find, and many of these paths may actually lead to feasible plans. Although the problem can be very search-intensive, the search arises from examining many different feasible plans in order to try to find a *good* plan.
- HTN planners do not have to do backward chaining in order to do goal-directed search. In an HTN planning system, the goal is specified as a task or a task network. The planner will not consider every planning operator that is applicable to the current state, but only those that occur in HTN methods that apply to the task it is trying to accomplish. Often only one or two methods are applicable—the rest get pruned.

Partial-Order Planning

In a planning problem where there are several goals to achieve, a backward-chaining total-order planner such as STRIPS (Nilsson 1980) will plan for the individual goals one at a time, developing a complete plan for each one before going on to the next. This approach has difficulty with *deleted-condition interactions* such as the well known Sussman anomaly (Waldinger 1990).

The discovery of deleted-condition interactions has sometimes led researchers to conclude that unless a planning has explicit mechanisms for detecting and handling deleted-condition interactions, the planner is incomplete (i.e., it will sometimes fail to find a plan when in fact a plan exists). However, this conclusion is incorrect. Although STRIPS's total-order backward-chaining algorithm is incomplete, total-order forward chaining is complete: a breadth-first search that does total-order forward chaining will generate every possible plan for a problem. Thus, as shown by Gupta and Nau (1992), a simple total-order forward-chaining strategy has no difficulty with deleted-condition interactions, even though it has no explicit way of detecting and handling these interactions.

Another use of partial-order planning is to prevent excessive backtracking during the planning process, by avoiding premature commitments to the order of the steps in a plan. However, in the planning domains discussed in this paper, avoiding commitments to the order of the steps would not prevent much backtracking, primarily because there often is only one logical order in which to perform the steps.

Difficulty with Backward Chaining and Partial-Order Planning

The previous sections have shown that the reasons normally used to motivate the use of backward chaining and partial-order planning do not necessarily apply to HTN planning in real-world domains. In this section, we argue that in some real-world domains, backward chaining and partial-order planning can cause substantial difficulty. This derives primarily from that fact that if a planner is searching backward from the goal or is developing a partial-order plan, it often does not know the input state for each of its planning operators, which makes it difficult to reason about those planning operators. For example, in Figure 3, we cannot tell whether or not Step s_3 is executable unless we know whether Step s_2 comes before Step s_3, whether it comes before Step s_1, whether $z=y$, and whether $x=y$.

The standard way for AI planning systems to handle this problem is by requiring each planning operator to have its constraints and effects represented as lists of logical atoms, so that the planner can partially instantiate those atoms based on what things it *does* know. Such a representation makes planning possible, provided that the planning operators can represented in this fashion—but it makes it very difficult for the planner to do a number of things that may be important for realistic planning:

- doing complex numeric calculations (e.g., computing probabilities, kinematic calculations, or monetary calculations);
- interacting with external information sources (e.g., sensors, image analysis programs, or CAD modelers);

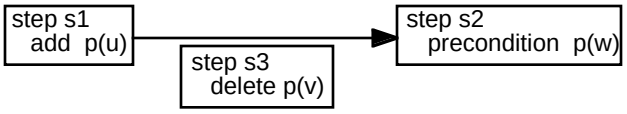


Figure 3. A partially-ordered plan.

- dealing with imperfect-information domains or multi-agent domains (such as contract bridge, negotiations, or military campaign planning).

Because the Bridge Baron and EDAPS expand tasks in the same order in which those tasks will be performed when the plan executes, this means that when they plan for each task, they already know as much as it is possible to know about what the state of the world will be at the time that the task will be performed. Consequently, the preconditions of each method could be written as arbitrary computer code, rather than as stylized logical expressions. This makes it feasible for EDAPS to make queries to a CAD modeler while developing a manufacturing plan, or for the Bridge Baron to do the complex numeric computations needed for reasoning about the probable locations of the opponents' cards in contract bridge.

For another example, consider finessing. This is a bridge tacticin which the declarer tries to win a trick with a high card, even though the opponents have a higher card. Depending on what cards the opponents have in their hands and how they choose to play them, the declarer may need to do different things as the finesse progresses; and to handle this, the Bridge Baron contains a table of *twenty-six* finesse situations. A partial-order planner might try using a finesse method while leaving some uninstantiated variables in the finesse method's preconditions, and then later in planning try to achieve these preconditions through the play of tricks earlier in the deal. Such a planner would have to decide which of the twenty-six situations to try to achieve—and it would not be immediately obvious which of them were even *possible* to achieve.

In contrast, because the Bridge Baron uses total-order planning to plan its declarer play, it would not try to apply any of the finesse methods until it had developed a partial plan resulting in a finesse situation. Furthermore, if it did generate a situation, it could easily check (using arbitrary computer code) whether the situation is an instance of one of the twenty-six different possible finesse situations—and then apply a finesse method, or not, as appropriate.

Theoretical Considerations

In STRIPS-style planning, total versus partial-order planning is an algorithmic issue; it does not affect the set of planning problems that can be represented or solved using STRIPS-style planning operators. However, the same is not true of HTN planning. As we explain below, only a proper subset of HTN planning problems can be represented if we force every method and the initial task network to be totally ordered. We also present a commitment strategy that will keep the task networks *mostly* totally-ordered without compromising the soundness or completeness of the planner.

Expressivity

Given a partially-ordered task network containing non-primitive tasks, in order to enforce the total-order restriction, one can consider enumerating all possible total orderings consistent with the partial order. However, this enumeration does not cover the whole set of candidate solutions for the original task network, because it precludes the possibility of interleaving subtasks of the non-primitive tasks in the task network. Below is an intuitive example:

Suppose you want to see a movie and have popcorn while you are doing so.

- The primitive tasks are Buy-Movie-Ticket, Buy-Popcorn, and Find-Seat; these tasks must be performed in this order (because you can only buy popcorn at the concession stand, which you can only get to if you have a ticket; and you certainly can't buy popcorn after you've found your seat.)
- The complex task is Watch-Movie, which decomposes into Buy-Movie-Ticket and Find-Seat.

An HTN planner starts out with the tasks Buy-Popcorn and Watch-Movie. A total-order planner must impose some ordering on these tasks, but neither ordering works: Buy-Popcorn before Watch-Movie leads to the primitive task sequence Buy-Popcorn, Buy-Movie-Ticket, and Find-Seat, but Buy-Movie-Ticket must be performed before Buy-Popcorn; and Watch-Movie before Buy-Popcorn leads to the primitive task sequence Buy-Movie-Ticket, Find-Seat, and Buy-Popcorn, but Buy-Popcorn must be performed before Find-Seat.

In contrast, a partial-order planner need not impose some ordering on Buy-Popcorn and Watch-Movie. It can start by decomposing Watch-Movie into Buy-Movie-Ticket and Find-Seat, and then interleaving Buy-Popcorn between Buy-Movie-Ticket and Find-Seat to produce a final plan. Thus, the expressivity of a partial-order HTN planning is stronger than the expressivity of a total-order HTN planning. This assertion can be proved using the definitions of complexity-based expressivity, model-theoretic expressivity, and operational expressivity developed by Erol *et al.* (1994b).

As reported in Erol *et al.* (1994b), the complexity of totally-ordered HTN planning is EXPSPACE-hard, and in DOUBLE-EXPTIME. In comparison, the complexity of HTN planning without this restriction is semi-decidable. Thus, from the computational-complexity perspective, totally-ordered HTN planning is significantly less expressive. Total-order HTN planning precludes the possibility of interleaving subtasks from different non-primitive tasks, thus eliminating inter-task interactions to a large extend. As a result, specialized algorithms that make use of this extra information can be developed. Note that the total-order restriction places HTN planning at the same complexity level as STRIPS-style planning, which is EXPSPACE-complete.

From the perspective of operational and model-theoretic expressivity, total-order HTN planning lies strictly between STRIPS-style planning and unrestricted HTN planning. Any problem represented in STRIPS language can be mapped to a total-order HTN planning problem. One such mapping is presented in (Erol et.al. 1994b). Obviously any total-order HTN planning problem can be represented by

the unrestricted HTN language. On the other hand, total-order HTN planning is not as expressive as the unrestricted HTN language: The set of plans for any total-order HTN problem is a context-free set. The set of solutions to unrestricted HTN planning problems may form arbitrary intersections of context-free sets, which properly contain context-free sets. Total-order HTN planning is strictly more expressive (with respect to operational and model theoretic expressivity) than the STRIPS language because the set of plans to any STRIPS-style planning problem form a regular set, which is properly contained in the set of context-free sets.

Increasing Expressivity by Relaxing the Total-Order Restriction

Our experience in practical planning applications indicates that most planning problems have tight ordering constraints. This feature can be utilized to develop specialized, efficient planning algorithms. However, some parts of a planning problem may not be totally-ordered, and we need to be able to deal with those problems without compromising soundness or completeness. As explained in the previous section, simply enumerating all total orderings consistent with the constraints in the task network would sacrifice completeness. In this section, we present a commitment strategy that will address this issue.

A commitment strategy, in the context of HTN planning, refers to the selection of the refinement strategy to be applied to the current task network. Erol *et al.* (1994b) presented a sound and complete HTN planning system called UMCP. UMCP's basic planning algorithm, which is based on refinement search, is as follows:

1. Input a planning problem $P=<d,I,D>$
2. Initialize OPEN-LIST to contain only d
3. if OPEN-LIST is empty then return "no solution"
4. Remove a task network *tn* from the OPEN-LIST
5. if *tn* is primitive, *tn*'s constraint formula is "TRUE"
   and all committed constraints have become true
   then return *tn* as a solution
6. Pick a refinement strategy *R* for *tn*
7. Apply *R* to *tn*. Insert the resulting set of task networks into OPEN-LIST
8. Goto Step 3.

The UMCP planner utilizes two refinement strategies (Step 6): Task decomposition refinement involves retrieving the set of methods associated with a non-primitive task in the task network, and then for each method generating a new task network by expanding the selected non-primitive task. Constraint enforcement refinement involves making the selected constraints necessarily true by restricting the possible values for variables and task orderings in the task network. Restrictions on possible values of variables and task orderings are stored in specialized data-structures, associated with each task network. Both of the refinement strategies are provably sound, complete, and systematic. As long as we give UMCP a sound and complete refinement strategy to use in Step 6, UMCP as a whole will be sound and complete. Below, we describe how to implement a sound and complete refinement strategy for UMCP that will explore earlier tasks first, while giving priority to ordering commitments, as well as variables that appear in earlier tasks.

Given a task network *tn*, let *tasks(tn)* denote the set of tasks in *tn*. Our modified refinement strategy for UMCP will construct a totally-ordered set of ground primitive tasks called *head(tn)*, such that these tasks are ordered to be before every other task in *tn*. Intuitively, *head(tn)* corresponds to a prefix of the final plan. In the initial task network input given to our modified version of UMCP, *head(tn)* is empty. Our refinement strategy will add ground primitive tasks to *head(tn)* as the planning proceeds. At the time that a solution is found, *head(tn)* will contain all the tasks in the final task network, and thus *head(tn)* will be the plan. Given a task network *tn*, here is how our refinement strategy will select what refinement to perform in Step 6 of UMCP:

1. If there are any constraints involving tasks in head(tn), then select constraint enforcement on them.
2. Let φ be the set of all tasks in *tasks(tn) – head(tn)* that are not necessarily ordered to come after any other tasks in *tasks(tn) – head(tn)*. If φ contains non-primitive tasks, then select task decomposition refinement on those tasks.
3. If φ consists of only primitive tasks, then for each ground linearization of those tasks in φ, generate a new task network where the ground linearization is appended to the *head(tn)*.

Refinement strategies 1 and 2 are from UMCP's original refinement strategies, and hence they are provably sound, complete, and systematic. These two refinement strategies are biased towards handling constraints and compound tasks that appear earlier in the task network.

Refinement strategy 3 enumerates all possible orderings and variable bindings for a set of primitive tasks in *tn*, which are already ordered before every other task in *tn* except for those in *tail(tn)*. Since those tasks are primitive, enumeration of their all possible ground linearizations preserves soundness, completeness, and systematicity. Note that this schema circumvents the problem with enumerating all possible orderings of the input task network *tn* right away: *tn* may contain non-primitive tasks, and ordering them would preclude the interleaving the subtasks of the non-primitive tasks.

Conclusions

AI planning is becoming increasing useful in practical planning problems—but the techniques that work well in practice are not always the ones that AI planning researchers might expect. For example, in developing successful HTN planning systems for two different practical problems, we found it better not to use the "traditional" techniques of backward chaining and partial-

order planning, but instead to develop a total-order planning control strategy that expands tasks in the order that they will be executed.

In this paper we have examined the assumptions behind backward chaining and partial-order planning, and have explained why these assumptions do not always reflect the characteristics of real-world planning problems. We have further explained how backward chaining and partial-order planning can sometimes be very difficult to use in practice, and how total-order HTN planning can overcome those difficulties.

We have also examined the expressive power of total-order HTN planning, and concluded that while it is strictly more expressive than planning with STRIPS-style operators, it is strictly less expressive than unrestricted HTN planning. We have proposed a way to relax the "total-order" restriction, to produce a sound and complete forward-search approach that has the same expressive power as unrestricted HTN planning. A promising topic for future work will be to implement this control strategy and try it out on a variety of planning problems.

Acknowledgements

This work was supported in part by an AT&T PhD scholarship to Stephen J. J. Smith; Maryland Industrial Partnerships (MIPS) Grant 501.15; Great Game Products; by NSF grants NSF EEC 94-02384, IRI-9306580, and DDM-9201779; ARPA grant DABT63-95-C-0037; ONR grant N000149610888; and in-kind contributions from Spatial Technologies and Bentley Systems. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the funders.

We also wish to thank James Hendler for many helpful discussions.

References

Chandrasekaran, R. 1997. Program for a better bridge game: A college partnership aids industry research. *The Washington Post*, Sept. 15, 1997. Washington Business section, pp. 1, 15, 19.

Erol, K.; Hendler, J.; and Nau, D. 1994. "UMCP: A Sound and Complete Procedure for Hierarchical Task-Network Planning," *Proc. 2nd Int'l Conf. on AI Planning Systems*, 249-254.

Erol, K.; Nau, D.; Hendler, J. 1994. "HTN Planning: Complexity and Expressivity." *Proc. AAAI-94*.

Erol, K.; Hendler, J.; and Nau, D. Complexity results for hierarchical task-network planning. *Annals of Mathematics and Artificial Intelligence* 18:69–93, 1996.

Great Game Products. 1997. *Bridge Baron.* <http://www.bridgebaron.com>.

Hebbar, K.; Smith, S.; Minis, I.; and Nau, D. 1996. "Plan-Based Evaluation of Designs for Microwave Modules," *ASME Computers in Engineering Conf.*

Korf, R. 1994. Best-first minimax search: Othello results. *AAAI-94*.

Gupta, N. and Nau, D. 1992. On the complexity of blocks-world planning. *Artificial Intelligenc*e 56(2-3), 223–254.

Nau, D.; Regli, W.; and Gupta, S. 1995. AI Planning Versus Manufacturing-Operation Planning: A Case Study. *IJCAI-95.*

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

Sacerdoti, E. D. 1977. *A Structure for Plans and Behavior.* American Elsevier Publishing Company.

Schaeffer, J. 1993. Presentation at plenary session, *AAAI Fall Symposium*.

Smith, S. J..; Nau, D. S.; and Throop, T. 1996. "A Planning Approach to Declarer Play in Contract Bridge", *Computational Intelligence*, 12(1), 106–130.

Smith, S. J.; Nau, D. S.; and Throop, T. 1996. "Total-Order Multi-Agent Task-Network Planning for Contract Bridge", *AAAI-96*, 108–113.

Smith, S. J.; Hebbar, K.; Nau, D.; and Minis, I. 1997. Integrating electrical and mechanical design and process planning. In Martti Mantyla, Susan Finger and Tetsuo Tomiyama (ed.), *Knowledge Intensive CAD, Volume 2*, pp. 269-288.

Smith, S. J.; Nau, D.; and Throop, T. 1998. Success in spades: using AI planning techniques to win the world championship of computer bridge. *IAAI-98.*

Tate, A. 1977. Generating project networks. *IJCAI-77*, 888–893.

Truscott, A. 1997. Bridge. *New York Times*, 16 August 1997, p. A19.

Tsuneto, R.; Erol, K.; Hendler, J.; and Nau, D. 1996. Commitment strategies in hierarchical task network planning. In *Proc. Thirteenth National Conference on Artificial Intelligence*, pp. 536-542.

Tsuneto, R.; Nau, D.; and Hendler, J. 1997. Plan-refinement strategies and search-space size. In *Proc. European Conference on AI Planning*.

Waldinger, R. 1990. Achieving several goals simultaneously. In J. Allen, J. Hendler, and A. Tate (ed.),*Readings in Planning*, Morgan Kaufmann, San Francisco, pp. 118-139.