AI Magazine Volume 19 Number 2 (1998) (© AAAI)

Computer Bridge

A Big Win for AI Planning

Stephen J. J. Smith, Dana Nau, and Tom Throop

■ A computer program that uses AI planning techniques is now the world champion computer program in the game of Contract Bridge. As reported in *The New York Times* and *The Washington Post*, this program—a new version of Great Game Products' BRIDGE BARON program—won the Baron Barclay World Bridge Computer Challenge, an international competition hosted in July 1997 by the American Contract Bridge League.

It is well known that the game tree search techniques used in computer programs for games such as Chess and Checkers work differently from how humans think about such games. In contrast, our new version of the BRIDGE BARON emulates the way in which a human might plan declarer play in Bridge by using an adaptation of hierarchical task network planning. This article gives an overview of the planning techniques that we have incorporated into the BRIDGE BARON and discusses what the program's victory signifies for research on AI planning and game playing.

One long-standing goal of AI research has been to build programs that play challenging games of strategy well. The classical approach used in AI programs for games of strategy is to do a game tree search using the well-known *minimax formula* (eq. 1) The minimax computation is basically a brute-force search: If implemented as formulated here, it would examine every node in the game tree. In practical implementations of minimax game tree searching, a number of techniques are used to improve the efficiency of this computation: putting a bound on the depth of the search, using alpha-beta pruning, doing transposition-table lookup, and so on. However, even with enhancements such as these, minimax computations often involve examining huge numbers of nodes in the game tree. For example, in the recent match between DEEP BLUE and Kasparov, DEEP BLUE examined roughly 60 billion nodes for each move (IBM 1997). In contrast, humans examine, at most, a few dozen board positions before deciding on their next moves (Biermann 1978).

Although computer programs have done well in games such as Chess and Checkers (table 1), they have not done as well in the game of Contract Bridge. Even the best Bridge programs can be beaten by the best players at many local Bridge clubs.

One reason why traditional game tree search techniques do not work well in Bridge is that Bridge is an imperfect-information game. Because Bridge players don't know what cards are in the other players' hands (except for, after the opening lead, what cards are in the dummy's hand), each player has only partial knowledge of the state of the world, the possible actions, and their effects. If we were to construct a game tree that included all the moves a player might be able to make, the size of this tree would vary depending on the particular Bridge deal—but it would include about 5.6×10^{44} leaf nodes in the worst case (Smith 1997, p. 226) and about 2.3×10^{24} leaf nodes in the average case (Lopatin 1992, p. 8). Because a Bridge hand is typically played in just a few minutes, there is not enough time for a game tree search to search enough of this tree to make good decisions.

Our approach to this problem (Smith 1997; Smith, Nau, and Throop 1996a, 1996b, 1996c) grows out of the observation that Bridge is a game of planning. The Bridge literature describes a number of tactical schemes (finessing, ruffing, cross-ruffing, and so on) that people combine into strategic 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 hierarchical task network (HTN) planning. We have

$$\text{minimax}(p) = \begin{cases} \text{our payoff at the node } p & \text{if } p \text{ is a terminal node} \\ \max\{\text{minimax}(q) : q \text{ is a child of } p\} & \text{if it is our move at the node } p \\ \min\{\text{minimax}(q) : q \text{ is a child of } p\} & \text{if it is our opponent's move at the node } p \end{cases}$$

Equation 1. Minimax Formula.

Connect Four	Solved
Go-Moku	Solved
Qubic	Solved
Nine-Men's Morris	Solved
Othello	Probably better than any human
Checkers	Better than any living human
Backgammon	Better than all but about 10 humans
Chess	Better than all but about 250 humans, possibly better
Scrabble	Worse than best humans
Go	Worst than best human 9 year olds
Bridge	Worse than the best players at many local clubs

Table 1. Computer Programs in Games of Strategy.
This is an updated and expanded version of similar tables from Schaeffer (1993) and Korf (1994).

	Brute-Force Search	**Our Approach**
Worst case	About 5.6×10^{44} leaf nodes	About 305,000 leaf nodes
Average case	About 2.3×10^{24} leaf nodes	About 26,000 leaf nodes

Table 2. Game Tree Size Produced in Bridge by a Full Game Tree Search and Our Hierarchical Task Network Planning Approach.

developed an algorithm for declarer play in Bridge that uses planning techniques to develop game trees whose size depends on the number of different strategies that a player might pursue rather than the number of different possible ways to play the cards. Because the number of sensible strategies is usually much less than the number of possible card plays, we are able to develop game trees that are small enough to be searched completely, as shown in table 2.

The remainder of this article contains overviews of the game of Bridge and HTN planning, a discussion of how we adapted HTN planning for declarer play in Bridge, a synopsis of related work by others, a discussion of how our program won the 1997 World Bridge Computer Challenge, a few notes about the application of our HTN planning techniques in other domains, and a discussion of what these things might signify for future work on both computer Bridge and AI planning.

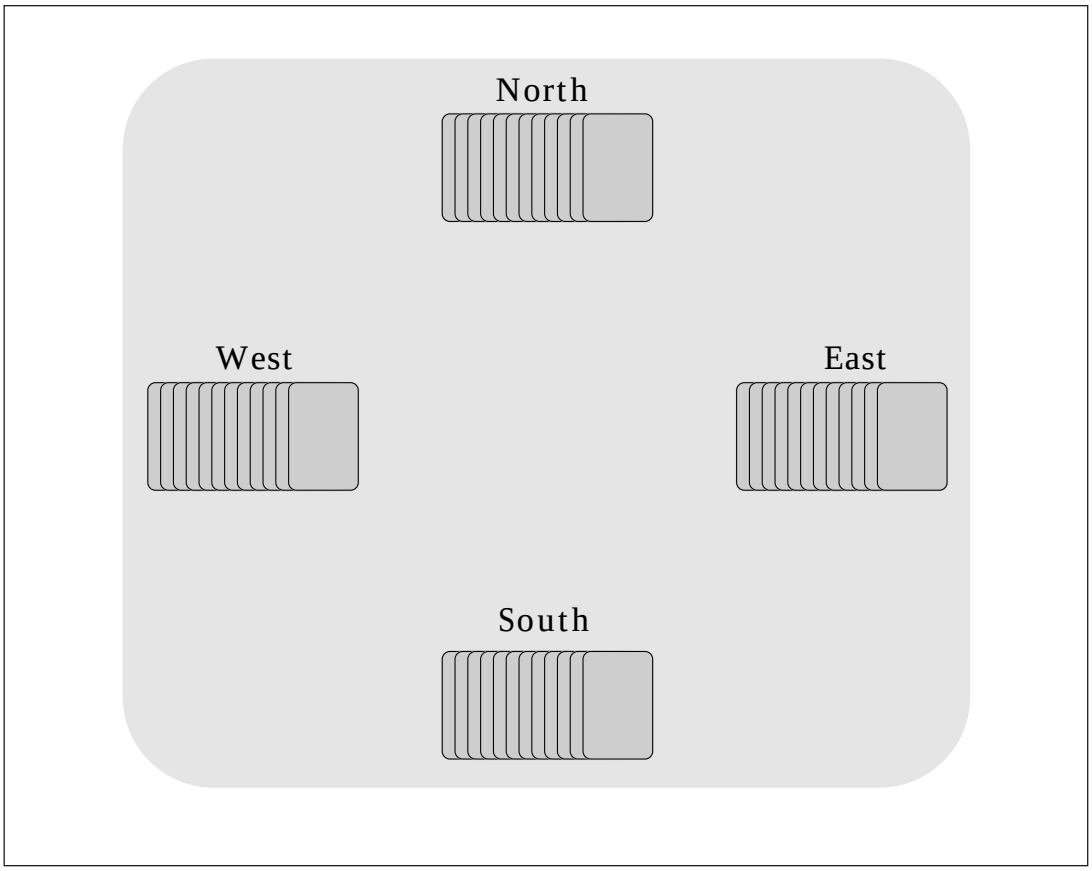


Figure 1. At the Beginning of a Bridge Hand, the 52 Cards Are Dealt Equally among the Four Players.

Overview of Bridge

Bridge is a game played by four players, using a standard deck of 52 playing cards divided into four *suits* (spades ♠, hearts ♡, diamonds ♢, and clubs ♣), each containing 13 cards. The players (who are normally referred to as North, South, East, and West) play as two opposing teams, with North and South playing as partners against East and West. A Bridge deal consists of two phases: (1) bidding and (2) play.

Bidding

Whichever player was designated as dealer for the deal deals the cards, distributing them equally among the four players, as shown in figure 1. Each player holds his/her cards so that no other player can see them.

The players make bids for the privilege of determining which suit is trump and what the level of the contract is. Nominally, each bid consists of two things: (1) some number of tricks that the bidder promises to take and (2) the suit that the bidder is proposing as the trump suit. However, various bidding conventions have been developed in which these bids are also used to convey information to the bidder's partner about how strong the bidder's hand is.

The bidding proceeds until no player wants to make a higher bid. At this point, the highest bid becomes the *contract* for the hand. In the highest bidder's team, the player who bid this suit first becomes the *declarer*, and the declarer's partner becomes the *dummy*. The other two players become the *defenders*.

Play

The first time that it is the dummy's turn to play a card, the dummy lays his/her cards on the table face up so that everyone can see them; during the card play, the declarer plays both the declarer's cards and the dummy's cards.

The basic unit of card play is the *trick*, in which each player in turn plays a card by placing it face up on the table, as shown in figure 2. The first card played is the card that was led, and whenever possible, the players must *follow suit*, that is, play cards in the suit of the card that was led. The trick is taken by whoever played the highest card in the suit led, unless some player plays a card in the trump suit, in

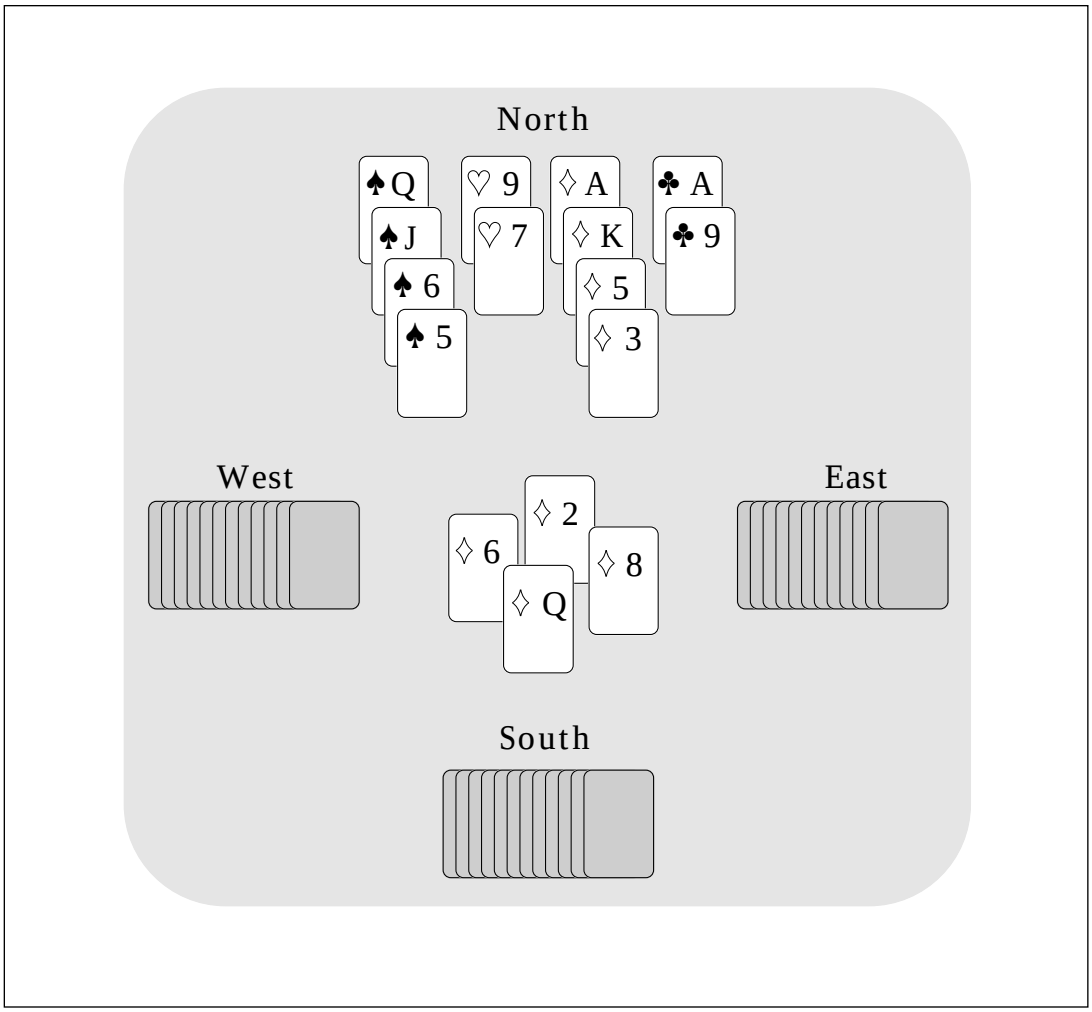


Figure 2. The Basic Unit of Play Is the Trick, in Which Each Player Places a Card Face Up in the Middle of the Table.

In this example, West leads the 6 of Diamonds, North (dummy) plays the 2 of Diamonds, East plays the 8 of Diamonds, and South (declarer) plays the Queen of Diamonds.

which case whoever played the highest trump card wins the trick.

The card play proceeds, one trick at a time, until no player has any cards left. At this point, the Bridge hand is scored according to how many tricks each team took and whether the declarer's team took as many tricks as it promised to take during the bidding.

In playing the cards, there are a number of standard tactical ploys that players can use to try to win tricks. These ploys have standard names (such as ruffing, cross-ruffing, finessing, cashing out, and discovery plays), and the ability of a Bridge player depends partly on how skillfully he/she can plan and execute these ploys. The importance of these ploys is especially true for the declarer, who is responsible for playing both the declarer's cards and the dummy's cards. In most Bridge hands, the declarer will spend some time at the beginning of the game formulating a strategic plan for how to play the declarer's cards and the dummy's cards. This plan will typically be some combination of various tactical ploys. Because of the declarer's uncertainty about what cards are in the opponents' hands and how the opponents might choose to play these cards, the plan will usually need to contain contingencies for various possible card plays by the opponents.

Overview of Hierarchical Task Network Planning

HTN planning (Wilkins 1988; Currie and Tate 1985; Sacerdoti 1977; Tate 1977) is an AI planning methodology that creates plans by task decomposition. *Task decomposition* is a process

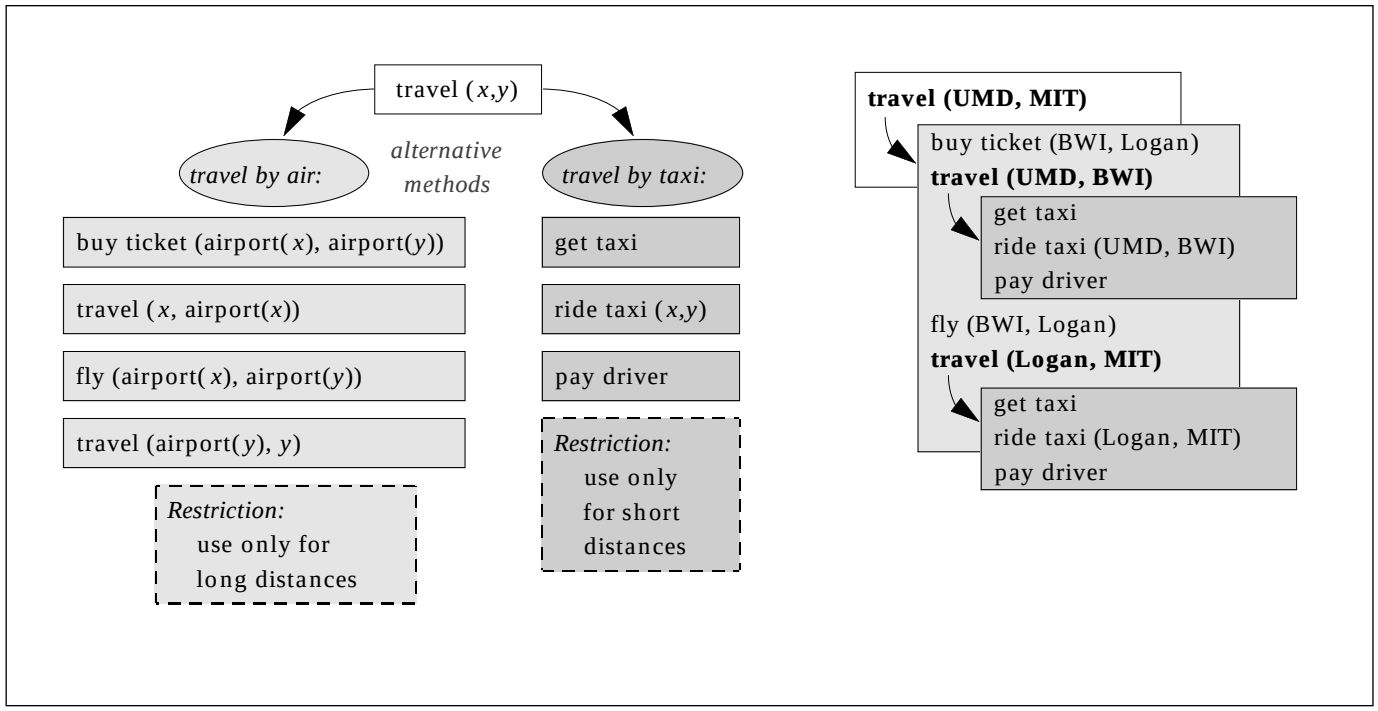


Figure 3. Two Methods for Traveling from One Location to Another and How They Might Be Applied to the Task of Traveling from the University of Maryland to the Massachusetts Institute of Technology.

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. Each *method* includes a prescription for how to decompose some task into a set of subtasks, with various restrictions that must be satisfied 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. 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.

Figure 3 shows two methods for the task of traveling from one location to another: (1) traveling by air and (2) traveling by taxi. Traveling by air involves the subtasks of purchasing a plane ticket, traveling to the local airport, flying to an airport close to our destination, and traveling from there to our destination. Traveling by taxi involves the subtasks of calling a taxi, riding in it to the final destination, and paying the driver. Each method has restrictions on when it can be used: Air travel is only applicable for long distances, and travel by taxi is only applicable for short distances.

Now, consider the task of traveling from the University of Maryland to the Massachusetts Institute of Technology. Because this is a long distance, the travel by taxi method is not applicable; so, we must choose the travel by air method. As shown in figure 3, this method decomposes the task into the following subtasks: purchase a ticket from Baltimore-Washington International (BWI) airport to Logan airport, travel from the University of Maryland to BWI, fly from BWI to Logan, and travel from Logan to MIT. For the subtasks of traveling from the University of Maryland to BWI and traveling from Logan to MIT, we could use the travel by taxi method to produce additional subtasks, as shown in figure 3.

Solving a planning problem using HTN planning is generally much more complicated than in this simple example. Here are some of the complications that can arise:

First, the planner might need to recognize and resolve interactions among the subtasks. For example, in planning how to get to the airport, one needs to make sure one will arrive there in time to catch the plane.

Second, in the example in figure 3, it was always obvious which method to use, but in general, more than one method can be applicable to a task. If it is not possible to solve the

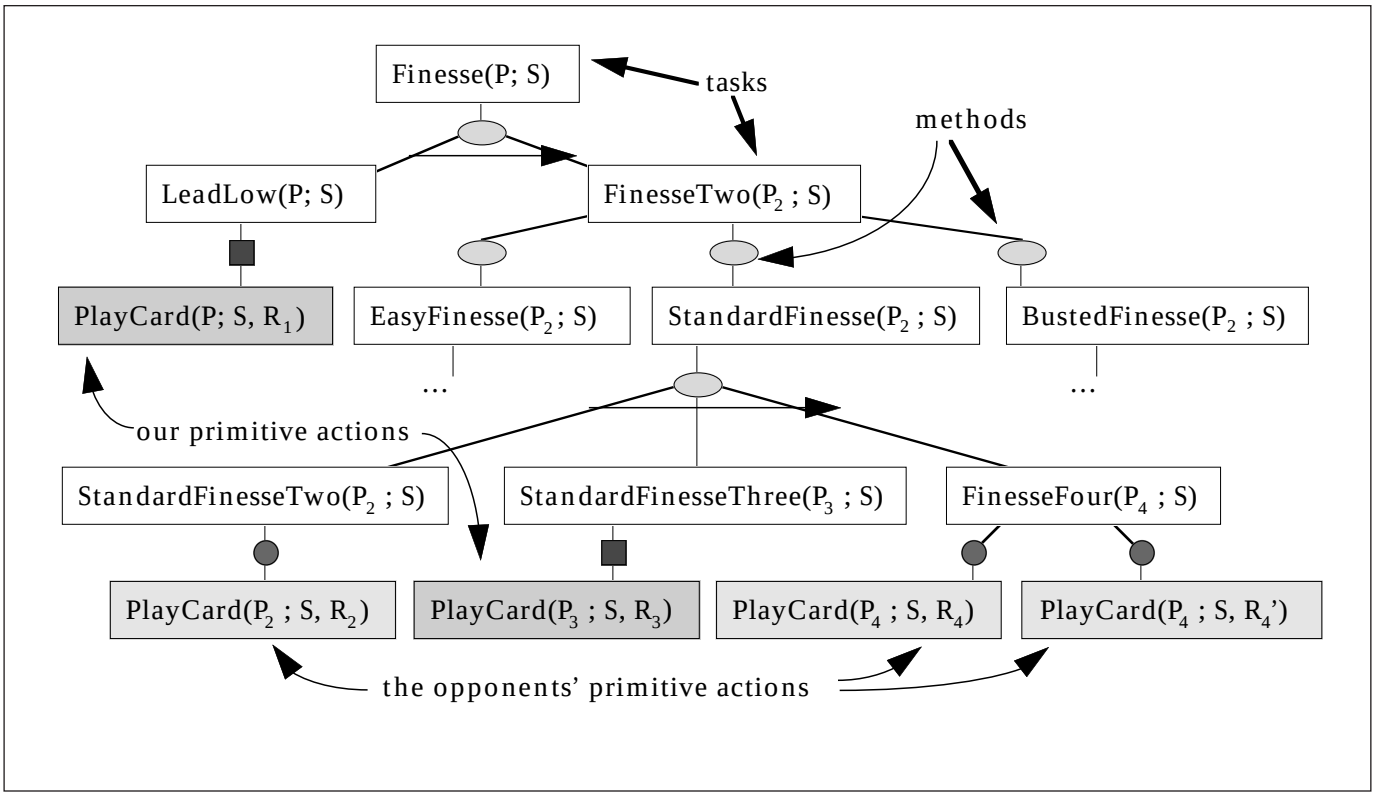


Figure 4. A Portion of TIGNUM 2's Task Network for Finessing.

subtasks produced by one method, it might be necessary to backtrack and try another method instead.

The first HTN planners were developed more than 20 years ago (Tate 1976; Sacerdoti 1974). However, because of the complicated nature of HTN planning, it was not until much later that researchers began to develop a coherent theoretical basis for HTN planning. A formal characterization of HTN planning now exists that shows it to be strictly more expressive than planning with STRIPS-style operators (Erol, Nau, and Hendler 1994), which has made it possible to establish a number of formal properties, such as soundness and completeness of planning algorithms (Erol, Hendler, and Nau 1994), complexity (Erol, Hendler, and Nau 1997), and the relative efficiency of various control strategies (Tsuneto, Nau, and Hendler 1996; Tsuneto et al. 1996). Domain-specific HTN planners have been developed for a number of industrial problems (Smith, Hebbar, et al. 1996; Smith, Nau, et al. 1996; Aarup et al. 1994; Wilkins and Desimone 1994), and a domain-independent HTN planner is available at www.cs.umd.edu/projects/plus/umcp/manual for use in experimental studies.

Our Adaptation of Hierarchical Task Network Planning for Bridge

We built a computer program called TIGNUM 2, which uses an adaptation of HTN planning techniques to plan declarer play in Contract Bridge. To represent the various tactical schemes of card playing in Bridge, TIGNUM 2 uses structures similar to HTN methods but modified to represent multiagency and uncertainty. TIGNUM 2 uses *state information sets* to represent the locations of cards that the declarer is certain of and *belief functions* to represent the probabilities associated with the locations of cards that the declarer is not certain of.

Some methods refer to actions performed by the opponents. In TIGNUM 2, we allow these methods to make assumptions about the cards in the opponents' hands and design our methods so that most of the likely states of the world are each covered by at least one method. In any of our methods, the subtasks are totally ordered; that is, the order in which the subtasks are listed for a method is the order in which these subtasks must be completed. For example, figure 4 shows a portion of our task

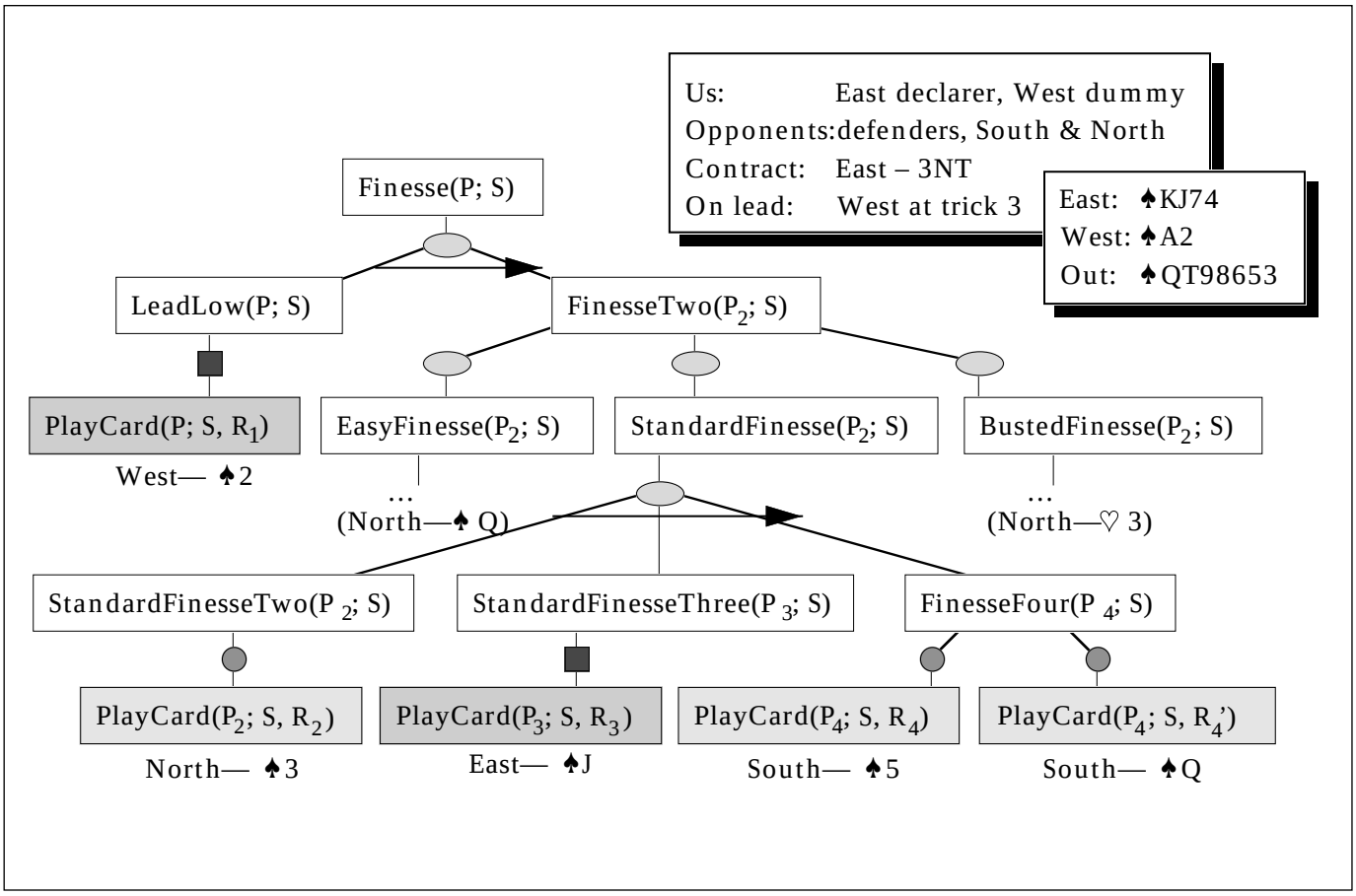


Figure 5. An Instantiation of the Finesse Method for a Specific Bridge Hand.

network for finessing in Bridge. Note that it refers to actions performed by each of the players in the game.

To generate game trees, our planning algorithm uses a procedure similar to task decomposition to build up a game tree whose branches represent moves generated by these methods. It applies all methods applicable to a given state of the world to produce new states of the world and continues recursively until there are no applicable methods that have not already been applied to the appropriate state of the world.

For example, figure 5 shows how our algorithm would instantiate the finessing method of figure 4 for a specific Bridge hand. In figure 5, West (the declarer) is trying a *finesse*, a tactical ploy in which a player tries to win a trick with a high card by playing it after an opponent who has a higher card. If North (a defender) has the ♠Q but does not play it when spades are led, then East (the dummy) will be able to win a trick with the ♠J because East plays after North. (North wouldn't play the ♠Q if he/she had any alternative because then East would win the trick with the ♠K and win a later trick with the ♠J.) However, if South (the other defender) has the ♠Q, South will play it after East plays the ♠J, and East will not win the trick.

Figure 6 shows the game tree resulting from the instantiation of the finessing method. This game tree is produced by taking the plays shown in figure 5 and listing them in the order in which they will occur. In figure 6, the declarer has a choice between the finessing method and the *cashing-out* method, in which the declarer simply plays all the high cards that are guaranteed to win tricks.

For a game tree generated in this manner, the number of branches from each state is not the number of moves that an agent can make (as in conventional game tree search procedures) but, instead, is the number of different tactical schemes that the agent can use. As shown in table 2, using the schemes as the branches results in a smaller branching factor and a much smaller search tree: Our planning algorithm generates game trees small enough that it can search them all the way to the end

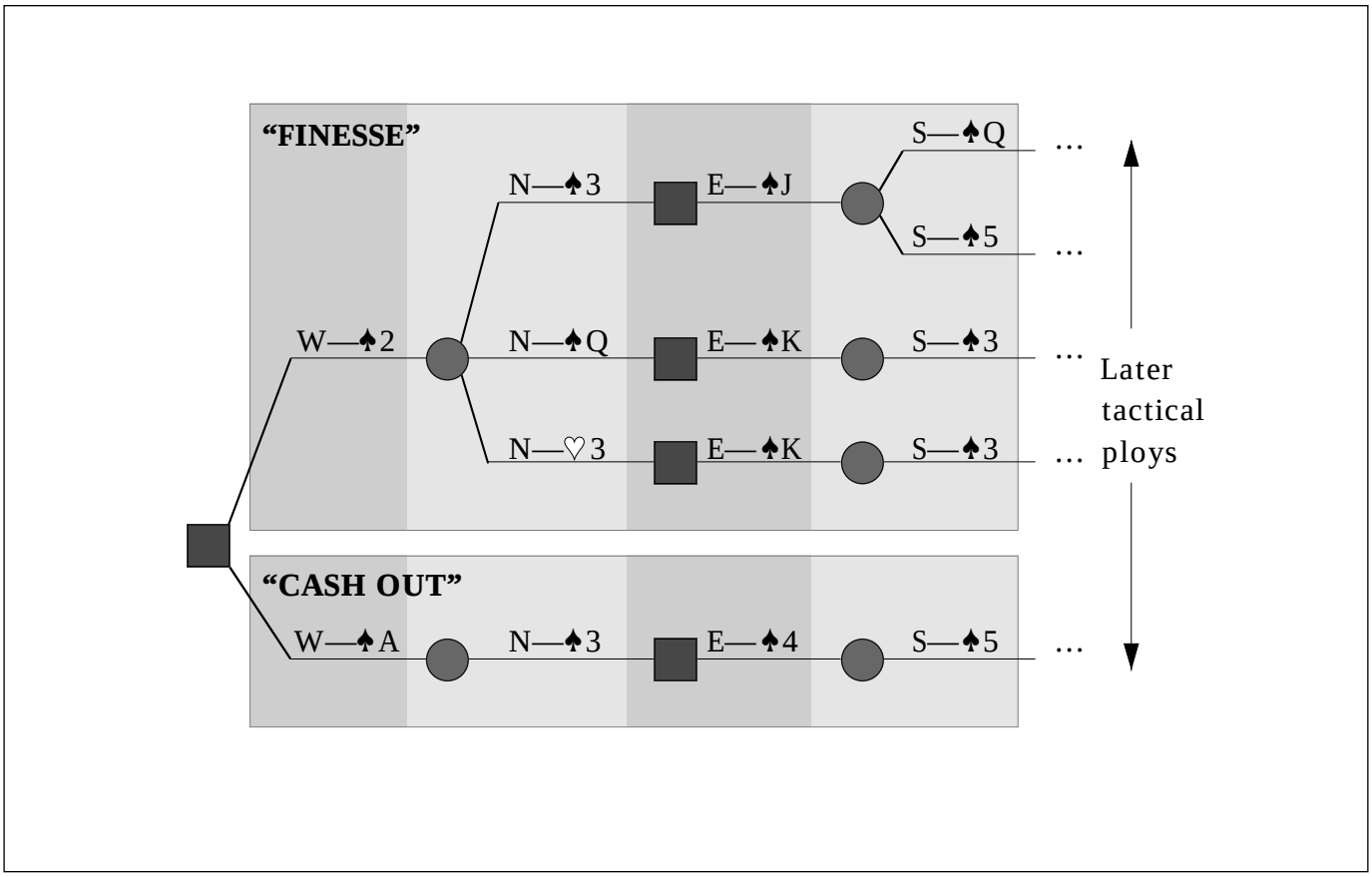


Figure 6. The Game Tree Produced by Instantiating the Finesse Method.

to predict the likely results of the various sequences of cards that the players might play.

To evaluate the game tree at nodes where it is the declarer's turn to play a card, our algorithm chooses the play that results in the highest score. For example, in figure 7, West chooses to play the ♠A that resulted from the cash-out method, which results in a score of +600, rather than the ♠2 that resulted from the finesse method, which results in a score of +270.46.

To evaluate the game tree at nodes where it is an opponent's turn to play a card, our algorithm takes a weighted average of the node's children, based on probabilities generated by our belief function. For example, because the probability is 0.9844 that North holds at least one *low* spade—that is, at least one spade other than the ♠Q—and because North is sure to play a low spade if North has one, our belief function generates the probability of 0.9844 for North's play of a low spade. North's other two possible plays are much less likely and receive much lower probabilities.

Implementation and Testing

To test our implementation of TIGNUM 2, we played it against Great Game Products' BRIDGE BARON program (Great Game Products 1997). BRIDGE BARON was originally developed in 1980 and has undergone many improvements since then (Throop 1983), and it is generally acknowledged to be the best commercially available program for the game of Bridge. At the time of our tests, it had won four international computer Bridge championships. In its review of seven commercially available Bridge-playing programs (Manley 1993), the American Contract Bridge League rated BRIDGE BARON to be the best of the seven and rated the skill of BRIDGE BARON to be the best of the five that do declarer play without "peeking" at the opponents' cards.

In Smith (1997), the results of our comparison of TIGNUM 2 declarer play against BRIDGE BARON's declarer play on 1000 randomly generated Bridge deals (including both suit and no-trump contracts). To do this comparison, we formed the following two teams: (1) the BRIDGE

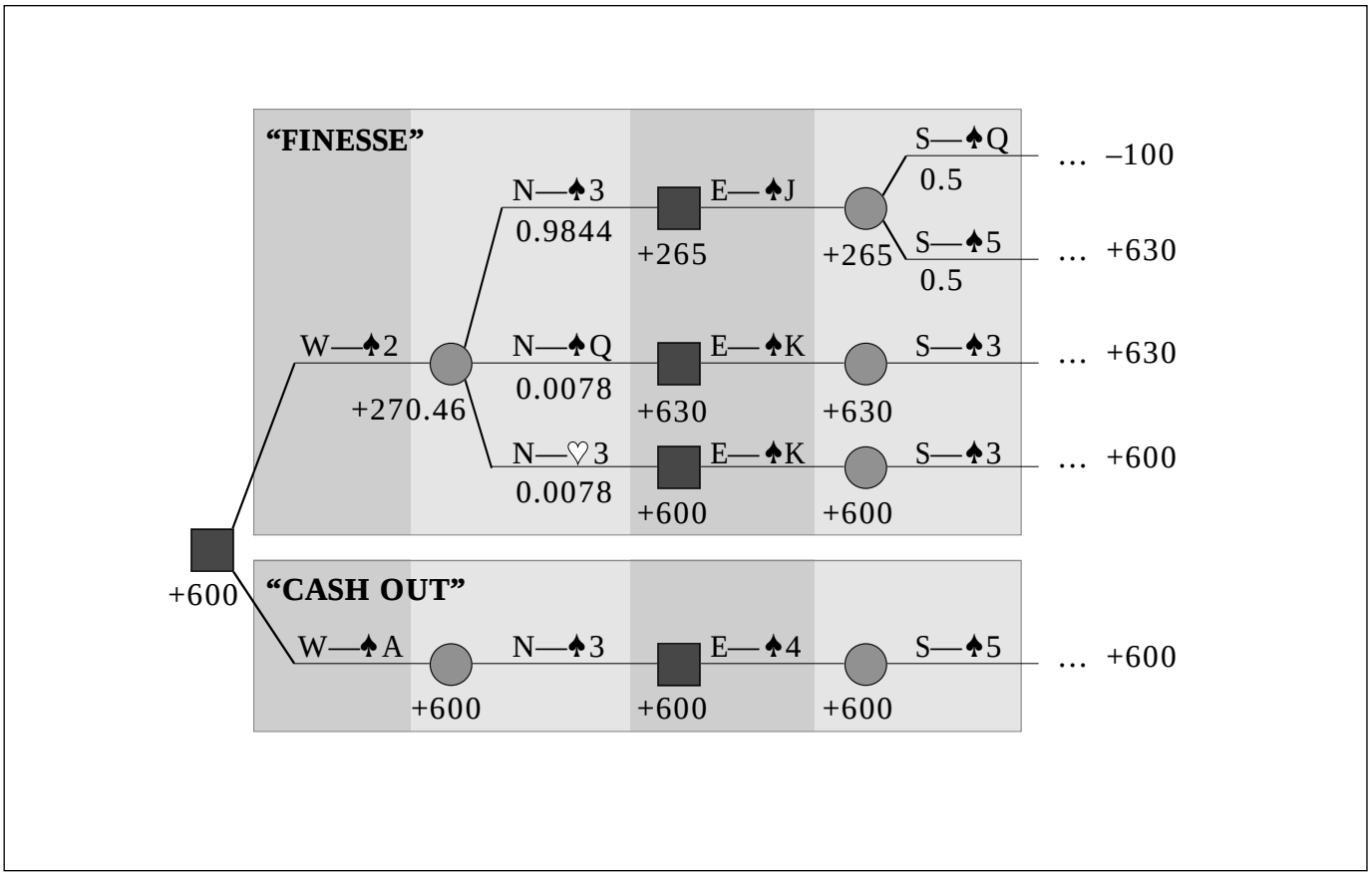


Figure 7. Evaluating the Game Tree of Figure 6.

BARON team, consisting of two copies of BRIDGE BARON and (2) the TIGNUM 2 team, consisting of two copies of the following combination of programs—TIGNUM 2 for use in declarer play and BRIDGE BARON for use in bidding and defender play (because TIGNUM 2 does not do bidding and defender play). To eliminate the possibility of either team gaining an advantage simply by the luck of the deal, each deal was played twice, once with the TIGNUM 2 team playing North and South and the BRIDGE BARON team playing East and West and once with the TIGNUM 2 team playing East and West and the BRIDGE BARON team playing North and South. To score each deal, we used *Swiss team board-a-match scoring,* in which the winner is defined to be the team getting the higher number of total points for the deal (if the teams have the same number of total points for a deal, then each team wins one-half the deal). In our comparison, the TIGNUM 2 team defeated the BRIDGE BARON team by 250 to 191, with 559 ties. These results are statistically significant at the $\alpha = 0.025$ level. We had never run TIGNUM 2 on any of these deals before this test, so these results are free from any training-set biases.

These tests were performed in February 1997. Since then, we have made additional changes to TIGNUM 2 that have improved its performance considerably. Furthermore, we have worked with Great Game Products to develop a new version of BRIDGE BARON, BRIDGE BARON 8, that uses the TIGNUM 2 code to plan its declarer play. The version of the TIGNUM 2 code used in BRIDGE BARON 8 contains 91 HTN task names and at least 400 HTN methods.

As described later in this article, a prerelease version of BRIDGE BARON 8 won the latest world-championship computer Bridge competition, and the official release went on sale in October 1997. In his review of Bridge programs, Jim Loy (1997) said of this new version of BRIDGE BARON: "The card play is noticeably stronger, making it the strongest program on the market."

Other Work on Computer Bridge

Most of the successful computer programs for Bridge other than ours are based on the use of domain-dependent pattern-matching tech-

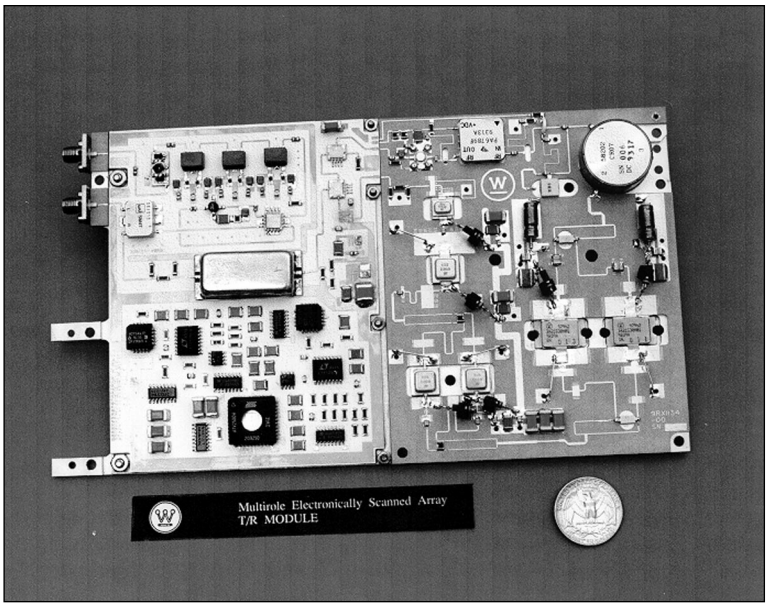


Figure 8. The Same Basic Approach (and Even Some of the Same Code!) That We Used for Planning Declarer Play in BRIDGE BARON *Is Also Used in a Progam That Plans How to Manufacture Microwave Transmit-Receive Modules (MWMs), Such as the One Shown Here.*

niques without much lookahead. However, several researchers have tried to develop Bridge programs that use adaptations of classical game tree search. Obviously, one of the biggest problems is how to handle the uncertainty about what cards are in the other players' hands. The most common approach to this problem—used, for example, late in the play in BRIDGE BARON—has been to use Monte Carlo techniques.[1] The basic idea is to generate many random hypotheses for how the cards might be distributed among the other players' hands, generate and search the game trees corresponding to each of the hypotheses, and average the results to determine the best move.

This Monte Carlo approach removes the necessity of representing uncertainty about the players' cards within the game tree itself, thereby reducing the size of the game tree by as much as a multiplicative factor of 5.2×10^6. However, the resulting game tree is still large, and thus, this method by itself has not been particularly successful.

Ginsberg (1996b) has developed a clever way to make the game tree even smaller. He starts with the Monte Carlo approach described previously, but to search each game tree, he uses a tree-pruning technique called *partition search*. Partition search reduces the branching factor of the game tree by combining similar branches—for example, if a player could play the ♠6 or the ♠5, partition search would generate only one branch for these two plays because they are basically equivalent. Like our approach, partition search produces game trees that are small enough to search completely. Ginsberg (1996a) has implemented this approach in a computer program called GIB, which was highlighted at the Fourteenth National Conference on Artificial Intelligence Hall of Champions Exhibit.

Frank and Basin (1995) have discovered a significant problem with Monte Carlo approaches: The use of Monte Carlo techniques introduces a fundamental incorrectness into the reasoning procedure. By this incorrectness, we do not mean the inaccuracies that might result from random variations in the game trees—inaccuracies because of random variation can be overcome (at the cost of additional computational effort) by generating a large enough sample of game trees. Rather, Frank and Basin have shown that any decision-making procedure that models an imperfect-information game as a collection of perfect-information games will be incapable of thinking about certain situations correctly.

We do not know how common such situations are, but they can be divided into several classes: The simplest one is what Bridge players call a *discovery play,* in which a player plays a card to observe what cards other players play in response, to gain information about what cards those players hold. A decision-making procedure that generates and searches random game trees as described previously will be unable to reason about discovery plays correctly because within each game tree, the procedure already thinks it has perfect information about the players' cards. We understand that Ginsberg is trying to develop a solution to this problem, but we have not yet seen a good way to overcome it.

Tournament Results

The most recent world-championship competition for computer Bridge programs was the Baron Barclay World Bridge Computer Challenge, which was hosted by the American Contract Bridge League. The five-day competition was held in Albuquerque, New Mexico, from 28 July to 1 August 1997. As reported in *The New York Times* (Truscott 1997) and *The Washington Post* (Chandrasekaran 1997), the winner of the competition was BRIDGE BARON—more specifically, a prerelease version of BRIDGE BARON 8, incorporating our TIGNUM 2 code. Here we describe the details of the competition.

The contenders included five computer programs: one from Germany, one from Japan,

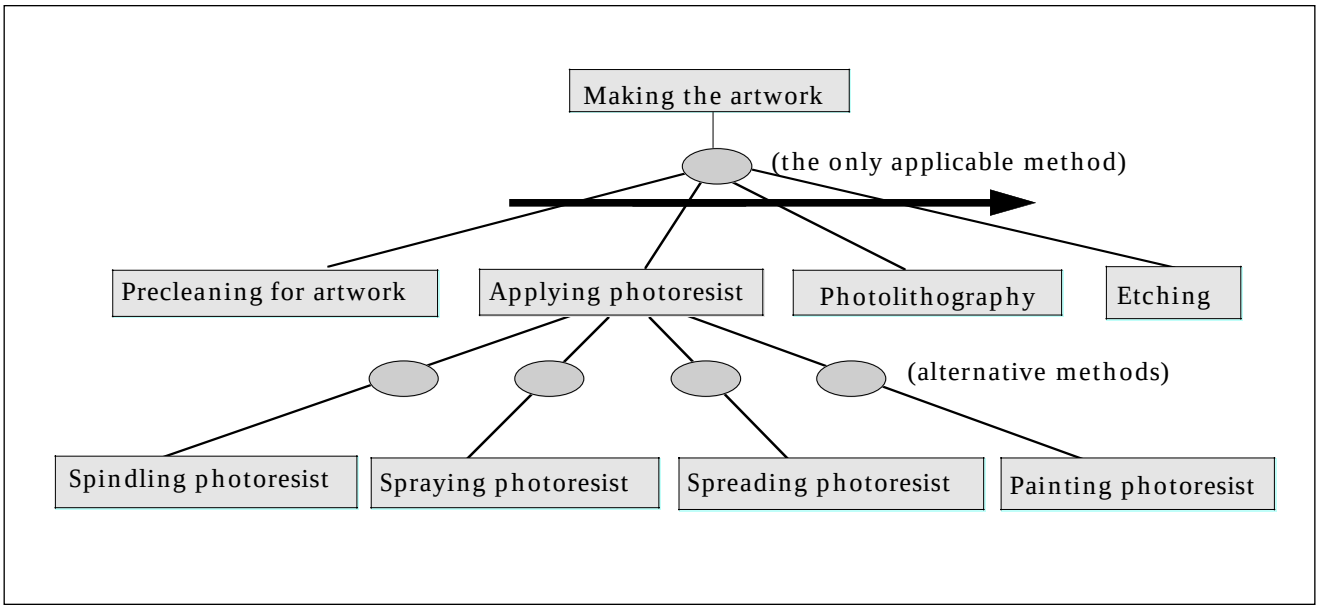


Figure 9. A Portion of a Task Network Containing Operations Used in Manufacturing Microwave Transmit-Receive Modules (MWMs).

and three from the United States. As far as we know, most of the programs were based on domain-dependent pattern-matching techniques that did not involve a lot of lookahead. The two exceptions included Ginsberg's GIB program and our new version of BRIDGE BARON. For the competition, two copies of each computer program competed together as a team, sitting East-West one of the times that a deal was played and North-South the other time the same deal was played.

The competition began with a qualifying round in which each of the five computer program teams played 10 matches against different human teams. Each match consisted of 4 deals; so, each program played a total of 40 deals. The results, which are shown in table 3, were scored using *international match points* (IMPs), a measure of how much better or worse a Bridge partnership's score is in comparison to its competitors. The bottom two programs (MEADOWLARK and GIB) were eliminated from the competition; the top three programs (Q-PLUS, MICROBRIDGE 8, and BRIDGE BARON) advanced to the semifinals.

In the semifinal match, Q-PLUS was given a bye, and MICROBRIDGE 8 played a head-to-head match against BRIDGE BARON. In this match, which consisted of 22 deals, BRIDGE BARON defeated MICROBRIDGE 8 by 60 IMPs, thus eliminating MICROBRIDGE 8 from the competition.

For the final match, BRIDGE BARON played a head-to-head match against Q-PLUS. In this match, which consisted of 32 deals, BRIDGE BARON defeated Q-PLUS by 22 IMPs, thus winning the competition. The final place of each program in the competition is shown in table 4.

Generality of Our Approach

To develop TIGNUM 2, we needed to extend HTN planning to include ways to represent and reason about possible actions by other agents (such as the opponents in a Bridge game) as well as uncertainty about the capabilities of these agents (for example, lack of knowledge about what cards they have). However, to accomplish this representation, we needed to restrict how TIGNUM 2 goes about constructing its plans. Most HTN planners develop plans in which the actions are partially ordered, postponing some of the decisions about the order in which the actions will be performed. In contrast, TIGNUM 2 is a total-order planner that expands tasks in left-to-right order.

Because TIGNUM 2 expands tasks in the same order that they will be performed when the plan executes, when it plans for each task, TIGNUM 2 already knows the state of the world (or as much as can be known about it in an imperfect-information game) at the time the task will be performed. Consequently, we can write each method's preconditions as arbitrary computer code rather than use the stylized logical expressions found in most AI planning systems. For example, by knowing the current state, TIGNUM 2 can decide which of 26 finesse

Program	Country	Score (IMPs)
Q-PLUS	Germany	+39.74
MICROBRIDGE 8	Japan	+18.00
BRIDGE BARON	United States	+7.89
MEADOWLARK	United States	–64.00
GIB	United States	–68.89

IMP = international match point.

Table 3. Results from the Qualifying Round of the Baron Barclay World Bridge Computer Challenge.
The top three contenders advanced to the semifinals; the bottom two contenders were eliminated from the competition.

Program	Country	Performance
BRIDGE BARON	United States	1st
Q-PLUS	Germany	2d
MICROBRIDGE 8	Japan	3d
MEADOWLARK	United States	4th
GIB	United States	5th

Table 4. The Final Place of Each Contender in the Baron Barclay World Bridge Computer Challenge.

situations are applicable: With partial-order planning, it would be much harder to decide which of them can be made applicable. The arbitrary computer code also enables us to encode the complex numeric computations needed for reasoning about the probable locations of the opponents' cards.

Because of the power and flexibility that this approach provides for representing planning operators and manipulating problem data, it can be useful in other planning domains that are different from computer Bridge. For example, consider the task of generating plans for how to manufacture complex electronic devices such as the microwave transmit-receive module (MWM) shown in figure 8. MWMs are complex electronic devices operating in the 1- to 20-gigahertz range that are used in radars and satellite communications. Designing and manufacturing electronic devices that operate in this frequency range is tricky because the placement of the components and the lengths and widths of the wires can change the electrical behavior of the devices.

Two of us have participated in the development of a system called EDAPS (electromechanical design and planning system) for computer-aided design (CAD) and manufacturing planning for MWMs (Smith 1997; Hebbar et al. 1996; Smith, Hebbar, et al. 1996; Smith, Nau, et al. 1996). In a contract with Northrop Grumman Corporation, our group is extending EDAPS into a tool to be used in Northrop Grumman's design and manufacturing facility in Baltimore, Maryland. EDAPS integrates commercial CAD systems for electronic design and mechanical design along with a program that generates manufacturing process plans for MWMs. The process-planning program uses the same basic approach (and even some of the same code!) that we used in our program for declarer play in Bridge. For example, figure 9 shows a portion of a task network for some of the operations used in MWM manufacture.

Conclusions

For games such as Chess and Checkers, the best computer programs are based on the use of brute- force game tree search techniques. In contrast, our new version of BRIDGE BARON bases its declarer play on the use of HTN planning techniques that more closely approximate how a human might plan the play of a Bridge hand.

Because computer programs still have far to go before they can compete

at the level of expert human Bridge players, it is difficult to say what approach will ultimately prove best for computer Bridge. However, BRIDGE BARON's championship performance in the Baron Barclay World Bridge Computer Challenge suggests that Bridge might be a game in which HTN planning techniques can be successful.

Furthermore, we believe that our work illustrates how AI planning is finally coming of age as a tool for practical planning problems. Other AI planning researchers have begun to develop practical applications of AI planning techniques in several other domains, such as marine oil spills (Agosta 1996), spacecraft assembly (Aarup et al. 1994), and military air campaigns (Wilkins and Desimone 1994). Furthermore, as discussed in the previous sections, the same adaptation of HTN planning that we used for computer Bridge is also proving useful for the generation and evaluation of manufacturing plans for MWMs. Because the same approach works well in domains that are as different as these, we are optimistic that it will be useful for a wider range of practical planning problems.

Acknowledgments

This work was supported in part by an AT&T Ph.D. scholarship to Stephen J. J. Smith, Maryland Industrial Partnerships grant 501.15; Advanced Research Projects Agency grant DABT 63-95-C-0037; and National Science Foundation grants NSF EEC 94-02384 and IRI-9306580. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the view of the funders.

Note

1. One exception is Lopatin's ALPHA BRIDGE program, which used classical game tree search techniques more or less unmodified, with a 20-ply (5-trick) search. ALPHA BRIDGE placed last at the 1992 Computer Olympiad.

References

Aarup, M.; Arentoft, M. M.; Parrod, Y.; Stader, J.; and Stokes, I. 1994. OPTIMUM-AIV: A Knowledge-Based Planning and Scheduling System for Spacecraft AIV. In *Intelligent Scheduling*, eds. M. Fox and M. Zweben, 451–469. San Francisco, Calif.: Morgan Kaufmann.

Agosta, J. M. 1996. Constraining Influence Diagram Structure by Generative Planning: An Application to the Optimization of Oil Spill Response. In Proceedings of the Twelfth Conference on Uncertainty in Artificial Intelligence, 11–19. Menlo Park, Calif.: AAAI Press.

Biermann, A. 1978. Theoretical Issues Related to Computer Game-Playing Programs. *Personal Computing* 2(9): 86–88.

Chandrasekaran, R. 1997. Program for a Better Bridge Game: A College Partnership Aids Industry Research. *The Washington Post*, Washington Business Section, 15 September, pp. 1, 15, 19.

Currie, K., and Tate, A. 1985. O-Plan–Control in the Open Planner Architecture. In Proceedings of the BCS Expert Systems Conference, 225–240. Cambridge, U.K.: Cambridge University Press.

Erol, K.; Hendler, J.; and Nau, D. 1997. Complexity Results for Hierarchical Task-Network Planning. *Annals of Mathematics and Artificial Intelligence* 18(1): 69–93.

Erol, K.; Hendler, J.; and Nau, D. 1994. UMCP: A Sound and Complete Procedure for Hierarchical Task-Network Planning. In *Proceedings of the Second International Conference on AI Planning Systems*, 249–254. Menlo Park, Calif.: AAAI Press.

Erol, K.; Nau, D.; and Hendler, J. 1994. HTN Planning: Complexity and Expressivity. In Proceedings of the Twelfth National Conference on Artificial Intelligence, 1123–1128. Menlo Park, Calif.: American Association for Artificial Intelligence.

Frank, I., and Basin, D. 1995. Search in Games with Incomplete Information: A Case Study Using Bridge Card Play, Research Paper, 780, Department of Artificial Intelligence, University of Edinburgh.

Ginsberg, M. 1996a. GIB, BRIDGE BARON, and Other Things. Usenet News Group REC.GAMES.BRIDGE, Message-ID 55aq9u$r31@pith.uroregon.edu, 31 October.

Ginsberg, M. 1996b. Partition Search. In Proceedings of the Thirteenth National Conference on Artificial Intelligence, 228–233. Menlo Park, Calif.: American Association for Artificial Intelligence.

Great Game Products. 1997. BRIDGE BARON. Available at www.bridgebaron.com, 28 December.

Hebbar, K.; Smith, S. J. J.; Minis, I.; and Nau, D. S. 1996. Plan-Based Evaluation of Design for Microwave Modules. In Proceedings of the ASME Design for Manufacturing Conference. Lubbock, Tex.: American Society of Mechanical Engineers.

IBM. 1997. How DEEP BLUE Works. Available at www.chess.ibm.com/meet/html/d.3.2.html, 28 December.

Korf, R. 1994. Best-First Minimax Search: OTHELLO Results. In Proceedings of the Twelfth National Conference on Artificial Intelligence, 1365–1370. Menlo Park, Calif.: American Association for Artificial Intelligence.

Lopatin, A. 1992. Two Combinatorial Problems in Programming Bridge Game. Computer Olympiad. Unpublished manuscript.

Loy, J. 1997. Review of Bridge Programs for PC Compatibles. Usenet News Group REC.GAMES.BRIDGE, Message-Id <343CAB0B.C6E7D2B1@pop.mcn.net>, 9 October.

Manley, B. 1993. Software "Judges" Rate Bridge-Playing Products. *The Bulletin of the American Contract Bridge League* 59(11): 51–54.

Sacerdoti, E. D. 1977. *A Structure for Plans and Behavior*. New York: American Elsevier.

Sacerdoti, E. D. 1974. Planning in a Hierarchy of Abstraction Spaces. *Artificial Intelligence* 5:115–135.

Schaeffer, J. 1993. Games: Planning and Learning. Presentation at plenary session of the AAAI Fall Symposium, 22–24 October, Research Triangle Park, North Carolina.

Smith, S. J. J. 1997. Task-Network Planning Using Total-Order Forward Search and Applications to Bridge and to Microwave Module Manufacture. Ph.D. diss., Department of Computer Science, University of Maryland at College Park.

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

Smith, S. J. J.; Nau, D. S.; and Throop, T. 1996b. AI Planning's Strong Suit. *IEEE Expert* 11(6): 4–5.

Smith, S. J. J.; Nau, D. S.; and Throop, T. 1996c. Total-Order Multiagent Task-Network Planning for Contract Bridge. In Proceedings of the Thirteenth National Conference on Artificial Intelligence, 108–113. Menlo Park, Calif.: American Association for Artificial Intelligence.

Smith, S. J. J.; Hebbar, K.; Nau, D. S.; and Minis, I. 1996. Integrated Electrical and Mechanical Design and Process Planning. Paper presented at the IFIP Knowledge Intensive CAD Workshop, 16–18 September, Pittsburgh, Pennsylvania.

Smith, S. J. J.; Nau, D. S.; Hebbar, K.; and Minis, I. 1996. Hierarchical Task-Network Planning for Process Planning for Manufacturing of Microwave Modules. In Proceedings of the Artificial Intelligence and Manufacturing Research Planning Workshop, 189–194. Menlo Park, Calif.: AAAI Press.

Tate, A. 1977. Generating Project Networks. In Proceedings of the Fifth International Joint Conference on Artificial Intelligence, 888–893. Menlo Park, Calif.: International Joint Conferences on Artificial Intelligence.

Tate, A. 1976. Project Planning Using a Hierarchic Nonlinear Planner, Technical Report, 25, Department of Artificial Intelligence, University of Edinburgh.

Throop, T. 1983. *Computer Bridge*. Rochelle Park, N.J.: Hayden.

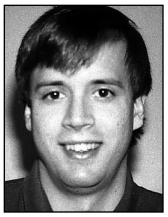
Truscott, A. 1997. Bridge. *New York Times*, 16 August 1997, p. A19.

Tsuneto, R.; Nau, D.; and Hendler, J. 1997. Plan-Refinement Strategies and Search-Space Size. In Proceedings of the European Conference on AI Planning, 24–26 September, Toulouse, France.

Tsuneto, R.; Erol, K.; Hendler, J.; and Nau, D. 1996. Commitment Strategies in Hierarchical Task Network Planning. In Proceedings of the Thirteenth National Conference on Artificial Intelligence, 536–542. Menlo Park, Calif.: American Association for Artificial Intelligence.

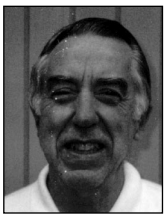
Wilkins, D. E. 1988. *Practical Planning*. San Francisco, Calif.: Morgan Kaufmann.

Wilkins, D. E., and Desimone, R. V. 1994. Applying an AI Planner to Military Operations Planning. In *Intelligent Scheduling*, eds. M. Fox and M. Zweben, 685–709. San Francisco, Calif.: Morgan Kaufmann.



Stephen J. J. Smith is an assistant professor of computer science at Hood College in Frederick, Maryland. He received a B.S. in computer science, mathematics, and Latin from Dickinson College and an M.S. and a Ph.D. in computer science from the University of Maryland at College Park. His e-mail address is sjsmith@nimue.hood.edu.

Dana Nau is a professor at the University of Maryland in the Department of Computer Science and the Institute for Systems Research (ISR), where he is coleader of the Systems Integration Research Group and the Electromechanical Systems Design and Planning Research Group and director of the Computer-Integrated Manufacturing Laboratory. He received his Ph.D. from Duke University in 1979, where he was a National Science Foundation (NSF) graduate fellow. He has more than 200 technical publications; for recent papers, see www.cs.umd.edu/users/nau. He has received an NSF Presidential Young Investigator Award, the ISR Outstanding Systems Engineering Faculty Award, and several best paper awards and is a AAAI fellow. His e-mail address is nau@cs.umd.edu.



Thomas A. Throop is the founder and president of Great Game Products. He received his B.S. in electrical engineering from Swarthmore College and did graduate work at Harvard University. He has devoted many years to research and development of computer algorithms to play Bridge. He created the first commercial version of the BRIDGE BARON program in 1985, and it has now won five World Computer Bridge Championships, with a winning record against each program it has played. His e-mail address is brbaron@erols.com.