IAAI-98/AAAI-98 Proceedings, pp. 1079–1086

Success in Spades: Using AI Planning Techniques to Win the World Championship of Computer Bridge

Stephen J. J. Smith **Dana S. Nau** **Thomas A. Throop**

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

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

Great Game Products
8804 Chalon Drive
Bethesda, MD, USA
brbaron@erols.com

Abstract

The latest world-championship competition for computer bridge programs was the *Baron Barclay World Bridge Computer Challenge*, hosted in July 1997 by the American Contract Bridge League. As reported in *The New York Times* and *The Washington Post*, the competition's winner was a new version of Great Game Products' *Bridge Baron* program. This version, Bridge Baron 8, has since gone on the market; and during the last three months of 1997 it was purchased by more than 1000 customers.

The Bridge Baron's success also represents a significant success for research on AI planning systems, because Bridge Baron 8 uses Hierarchical Task-Network (HTN) planning techniques to plan its declarer play. This paper gives an overview of those techniques and how they are used.

Problem Description

To be successful both in bridge competitions and as a commercial product, a computer program for the game of bridge must perform as well as possible in all aspects of the game. Customers want as challenging an opponent as possible, and of course the strongest program has the best chance of winning a competition. However, although researchers have had great success in developing high-performance programs for games such as chess and checkers, they have not had as much success in the game of contract bridge. Even the best bridge programs can be beaten by the best players at many local bridge clubs.

One important reason why it is difficult to develop good computer programs for bridge is that bridge is an imperfect information game. 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); thus each player has only partial knowledge of the state of the world, the possible actions, and their effects. If we were to use a naive adaptation of the classical game-tree search techniques used in computer programs for games such as chess and checkers, the game tree would need to include all of 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). Since a bridge hand is normally 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.

Until recently, the approach that was most successful in computer programs for the game of bridge was to use domain-dependent pattern-matching techniques that do not involve a lot of look-ahead. Most commercially available bridge programs use this approach, including the previous version of Great Game Products' *Bridge Baron* (Throop 1983, Great Game Products 1997). The Bridge Baron is generally acknowledged to be the best available commercial program for the game of contract bridge. Before the incorporation of AI techniques (described later in this paper), it had won four international computer bridge championships. In their review of seven commercially available bridge-playing programs (Manley 1993), the American Contract Bridge League rated the Bridge Baron to be the best of the seven, and rated the skill of the Bridge Baron to be the best of the five that do declarer play without "peeking" at the opponents' cards.

Despite the success of the Bridge Baron, improving its performance became more and more difficult as its relatively simple pattern-matching techniques revealed their limitations. Without more sophisticated techniques, the Bridge Baron would risk becoming less competitive with other bridge programs. Thus, we needed to develop more sophisticated AI techniques to improve the performance of the Bridge Baron.

Application Description

Overview of Bridge

Bridge is a game played by four players, using a standard deck of 52 playing cards, divided into four *suits* (spades ♠,

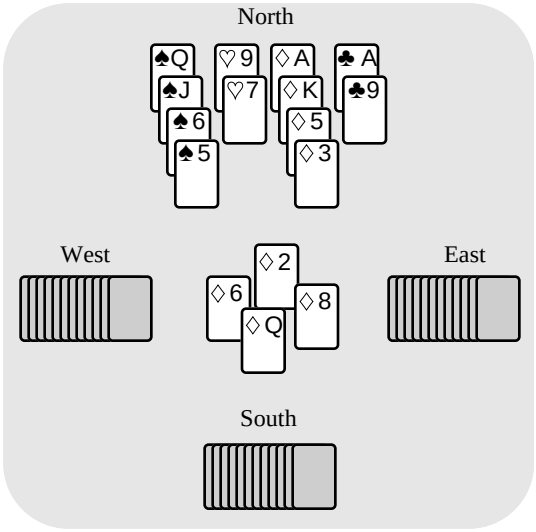

Figure 1. An example of a bridge hand during the play of the first trick. Here, South is declarer, North is dummy, and West and East are defenders.

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, *bidding* and *play*:

Bidding. The cards are dealt equally among the four players. 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 parts: some number of *tricks* (see below) that the bidder promises to take, and which suit 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 wishes to make a higher bid. At that point, the highest bid becomes the *contract* for the hand. In the highest bidder's team, the player who bid this suit first becomes *declarer*, and declarer's partner becomes *dummy.* The other two players become the *defenders.*

Play. The first time that it is dummy's turn to play a card (see below), dummy lays her or his cards on the table, face-up so that everyone can see them; and during the card play, declarer plays both declarer's cards and 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 1. The first card played is that 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 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 that point, the bridge hand is scored according to how many tricks each team took, and whether declarer's team took as many tricks as they promised to take during the bidding.

The Bridge Baron

The Bridge Baron consists of tens of thousands of lines of C code. Separate Bridge Baron executables run as a Windows application, an MS-DOS application, and a Macintosh application; the Windows application accounts for the overwhelming majority of sales.

In the Bridge Baron, over fifty thousand lines of code are devoted solely to the *bridge engine*, which calculates what bids to make and what cards to play. The bridge engine can be roughly divided into three parts, based on functionality: declarer play, defensive play, and bidding. The bridge engine represents knowledge largely through ad-hoc pieces of code developed to address particular bridge situations.

Other tens of thousands of lines of code handle the user interface. The user interface allows customers to play matches against the computer, to generate deals that conform to particular specifications, to see how the Bridge Baron determines its bids, and the like.

The user interface interacts with the bridge engine by calling C functions that recommend a particular bid or play in the current situation. The user interface also updates the data structures that provide the bridge engine with the details of the current situation.

Most commonly, a customer sits down with the Bridge Baron to play some number of deals. The customer takes the role of one of the four players, and the Bridge Baron takes the role of the customer's partner. The Bridge Baron also takes the roles of the customer's two opponents. Each role is separate: for example, one opponent does not know what cards the other opponent has, nor what cards the customer's partner has. (The customer can allow all of the Bridge Baron players to have complete knowledge of all of the cards by changing an option.)

For declarer play, previous versions of the Bridge Baron used ad-hoc pattern-matching techniques. However, Bridge Baron 8 (the latest version) makes use of the new AI planning techniques described later in this paper. Users may select whether they want to use the new AI planning techniques or the old ad-hoc techniques, which we did not remove from the Bridge Baron. We added options to limit the time that the new AI planning techniques spend on a particular play to 30 seconds, 60 seconds, or 120 seconds. This new version of the Bridge Baron became commercially available in October 1997.

Uses of AI Technology

To improve the play of the Bridge Baron, we supplemented its previously existing routines for declarer play with routines based on HTN (Hierarchical Task-Network) planning techniques. Our approach (Smith et al. 1996a;

Smith et al. 1996c; Smith et al. 1996e; Smith 1997) grew out of the observation that bridge is a game of planning. In playing the cards, there are a number of standard tactical ploys that may be used by the various players to try to win tricks. These 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 that player can plan and execute these ploys. This is especially true for declarer, who is responsible for playing both declarer's cards and dummy's cards. In most bridge hands, declarer will spend some time at the beginning of the game formulating a rough plan for how to play declarer's cards and dummy's cards. This plan will normally be some combination of various tactical ploys. Because of declarer's uncertainty about what cards are in the opponents' hands and how the opponents may choose to play those cards, the plan will usually need to contain contingencies for various possible card plays by the opponents.

We have taken advantage of the planning nature of bridge, by adapting and extending some ideas from HTN planning. We use planning techniques to develop game trees in which the number of branches at each node correspond to the different *strategies* that a player might pursue rather than the different cards the player might be able to play. Since the number of sensible strategies is usually much less than the number of possible card plays, this lets us develop game trees that are small enough to be searched completely, as shown in Table 1. Below we give an overview of HTN planning, and describe how we adapted it for use in the Bridge Baron.

Table 1. Game-tree size produced in bridge by a full game-tree search and by our HTN planning approach.

	Brute-force search	Our approach
Worst case	$\approx 5.6 \times 10^{44}$ leaves	≈ 305,000 leaves
Avg. case	$\approx 2.3 \times 10^{24}$ leaves	≈ 26,000 leaves

Overview of HTN Planning

HTN planning was originally developed more than 20 years ago (Sacerdoti 1974; Tate 1977), and has long been thought to have good potential for use in real-world planning problems (Currie and Tate 1985; Wilkins 1988), but it has only been recently that researchers have developed a coherent theoretical basis for HTN planning. Recent mathematical 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 a number of properties such as soundness and completeness of planning algorithms (Erol *et al.* 1994a), complexity (Erol *et al*. 1996), and the relative efficiency of various control strategies (Tsuneto *et al*. 1996; Tsuneto *et al*. 1997). A domain-independent HTN planner is available at <http://www.cs.umd.edu/projects/plus/umcp/manual> for use in experimental studies, and domain-specific HTN planners are being developed for several industrial problems (Aarup *et at*. 1994; Hebbar *et al.* 1996; Smith *et al*. 1996b; Smith *et al.* 1996d; Wilkins & Desimone 1994).

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*. Each method includes a prescription for how to decompose some task into a set of subtasks, 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. 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.

As a very simple example, Figure 2 shows two methods

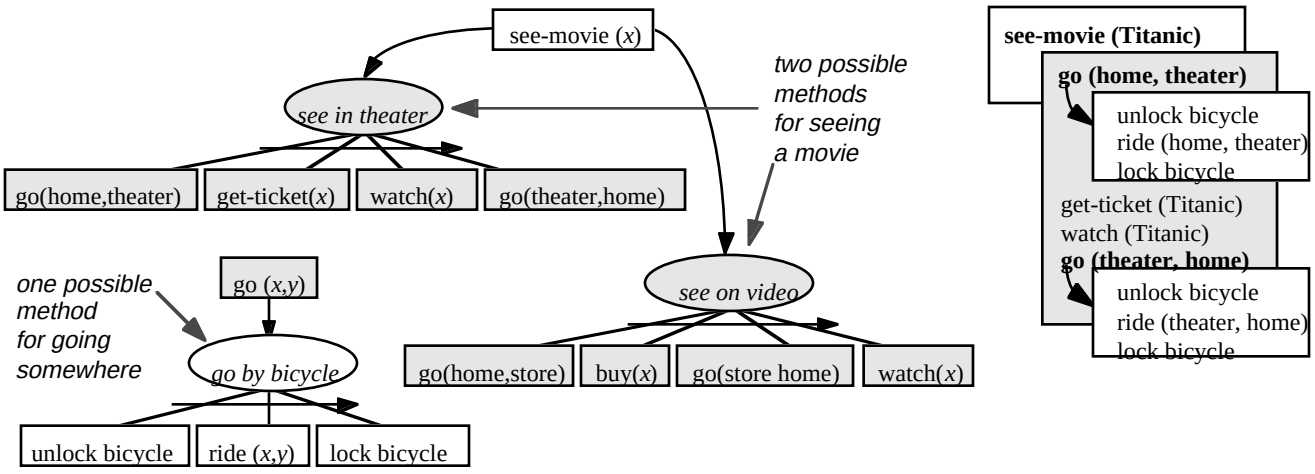

Figure 2. A very simple example illustrating how HTN planning might be used to plan to see a movie.

for seeing a movie: seeing it in a theater, and seeing it at home on videotape. Seeing it in a theater involves going to the theater, getting a ticket to the movie, watching the movie, and going home. Seeing it at home involves going to the store, buying the videotape, going home, and watching the movie. Figure 2 also shows one method for going places: riding a bicycle.

Now, consider the task of seeing the movie *Titanic*. Figure 2 shows how a planner might instantiate the "see in theater" method for this task, and how it might instantiate the "ride bicycle" method twice to handle the subtasks of getting to the theater and getting home.

Solving a planning problem using HTN planning is generally much more complicated than in this simple example. For example, the planner may need to recognize and resolve interactions among the subtasks (such as the necessity of getting to the theater before the movie begins). If such interactions cannot be worked out, then the planner may need to backtrack and try another method instead.

HTN Planning for Declarer Play

The *Tignum 2* portion of Bridge Baron 8 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 multi-agency and uncertainty. Tignum 2 uses *state information sets* to represent the locations of cards about which declarer is certain, and *belief functions* to represent the probabilities associated with the locations of cards about which declarer is not certain.

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 3 shows how our algorithm would instantiate some of its methods on a specific bridge hand. Here, South (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 West (a defender) has the ♡K, but does not play it when hearts are led, then North (dummy) will be able to win a trick with the ♡Q, because North plays after West. (West wouldn't play the ♡K if she or he had any alternative, because then North would win the trick with the ♡A and win a later trick with the ♡Q.) However, if East (the other defender) has the ♡K, East will play it after North plays the ♡Q, and North will not win the trick. Note that the methods refer 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 4 illustrates the evaluation of the game tree resulting from the instantiation of the finessing method. This game tree is produced by taking the plays shown in Figure 3 and listing them in the order in which they will occur. In Figure 4, declarer has a choice between the finessing method and the *cashing-out* method, in which declarer simply plays all of 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 the agent can employ. As shown in Table 1, this game tree is small enough that it can be searched all the

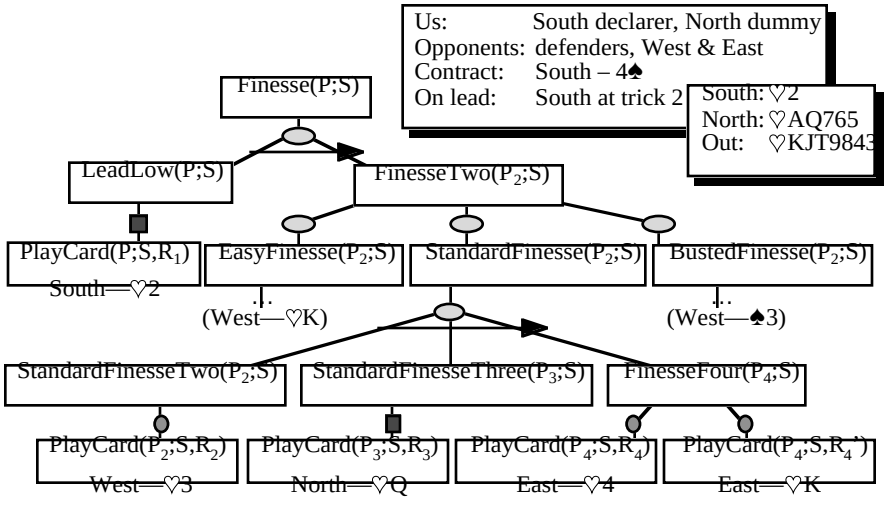

Figure 3: An instantiation of the "finesse" method for a specific bridge hand.

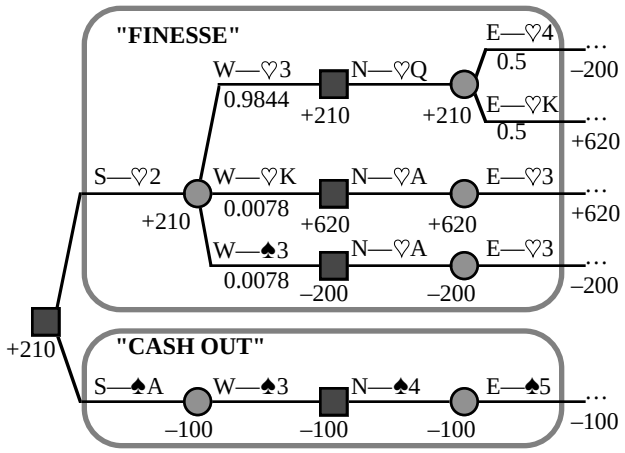

Figure 4: Evaluating the game tree produced from the instantiated "finesse" method of Figure 3.

way to the end, 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 declarer's turn to play a card, our algorithm chooses the play that results in the highest expected score. For example, in Figure 4, South chooses to play the ♡2 that resulted from the "finesse" method, which results in an expected score of +210, rather than the ♠A that resulted from the "cash out" method, which results in a score of -100.

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" heart—that is, at least one heart other than the ♡K—and because North is sure to play a low heart if North has one, our belief function generates the probability of 0.9844 for North's play of a low heart. North's other two possible plays are much less likely and receive much lower probabilities.

Application Use and Payoff

We brought a pre-release version of Bridge Baron 8 to the most recent world-championship competition for computer bridge programs: the *Baron Barclay World Bridge Computer Challenge*, which was hosted by the American Contract Bridge League (ACBL). The five-day competition was held in Albuquerque, New Mexico, from 28 July 1997 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 the Bridge Baron—more specifically, the winner was the pre-release version of Bridge Baron 8, incorporating our Tignum 2 code. The contenders included five computer programs; the final place of each program in the competition is shown in Table 2.

Table 2: 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

The official release of Bridge Baron 8 went on sale in October 1997; and during the last three months of 1997, more than 1000 customers purchased it. In his review of bridge programs, Jim Loy (1997) said of this new version of the Bridge Baron: "The card play is noticeably stronger, making it the strongest program on the market."

Application Development and Deployment

Chronology

A single graduate student did all of the programming of our HTN planning techniques for bridge, reusing a few hundred lines of code from the Bridge Baron. Other people—primarily two of them, with occasional assistance from at least two others—helped by discussing what knowledge to incorporate and how to perform the implementation. We did not use any formal development methods.

We began our work on adapting HTN planning techniques to bridge with a program called *Tignum*. We began working on some routines for reasoning about the probable locations of cards in 1989, most of which were eventually abandoned because they required too much execution time. We began work in earnest on Tignum in 1991, and after writing nine thousand lines of code, abandoned almost all of it in 1993.

We abandoned Tignum because it was poorly implemented. It did not allow us to consider alternative plays; it required every piece of bridge knowledge to be coded by hand with very little possibility of code reuse; it ran slowly; and it was difficult to maintain.

After abandoning Tignum in 1993, we began work immediately on Tignum 2, a much better implementation of the ideas used in Tignum. Tignum 2 allowed us to consider alternative plays. While bridge knowledge still had to be coded by hand, judiciously chosen macros and well-designed components made code reuse easy. Tignum 2 ran more quickly and was much easier to maintain.

Tignum 2 became ultimately successful in February 1997. To test our implementation of it, we played it against an older version of the Bridge Baron. In (Smith 1997) we reported the results of our comparison of Tignum 2 against this version of the Bridge Baron on 1,000 randomly generated bridge deals (including both suit and

no-trump contracts). Each deal was played twice, once with Tignum 2 as declarer and once with Bridge Baron as declarer; the winner of the deal was defined to be whichever declarer did better. On declarer play, Tignum 2 defeated Bridge Baron by 250 to 191, with 559 ties. These results are statistically significant at the a = 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 results allowed us to move forward with the incorporation of Tignum 2 into a new version of the Bridge Baron, *Bridge Baron 8*. In Bridge Baron 8, we added an option to allow customers to select whether they wanted to use the new AI planning techniques or the old ad-hoc techniques, which we did not remove from the Bridge Baron. We added options to limit the length of Tignum 2's planning time on a particular play to 30 seconds, 60 seconds, or 120 seconds. This new version of the Bridge Baron became commercially available in October 1997.

Lessons Learned

We learned several lessons from the development of this product. We wish we had thought out the implementation more carefully before beginning to write code for it. Having Tignum as an unintended prototype before the full implementation in Tignum 2 worked out surprisingly well; in the future, we intend to plan our development to include both a prototype and a full implementation.

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 those agents (for example, lack of knowledge about what cards they have). However, to accomplish this, 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.

Tignum 2 expands tasks in the same order that they will be performed when the plan executes, and so 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 that the task will be performed. Consequently, we can write each method's preconditions as arbitrary computer code, rather than using 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 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.

Maintenance

To date, only the original programmer has maintained the HTN planning routines in Bridge Baron 8, though we hope to have another programmer begin modifying these routines soon. In the past, the Bridge Baron has been improved on a daily basis, to make its performance of bidding and play better; in the future, we expect this to continue. Domain knowledge about bridge changes relatively infrequently, but plenty of domain knowledge about bridge has simply not been implemented yet in our HTN planning techniques.

The HTN planning routines in Bridge Baron 8 explicitly handle many bridge techniques: cashing out, ruffing out, crossing, finesses, free finesses, automatic finesses, marked finesses, proven finesses, sequence winners, length winners, winners that depend on splits, opponents on lead, opponents finessing against declarer and dummy, dangerous opponents, ducking, hold-up plays, discarding worthless cards, drawing trumps, ruffing, and setting up ruffs. Some obvious domain knowledge missing from these routines include endplays and squeezes; we have not handled these techniques because they are relatively rare. As well, the existing techniques certainly need to be improved.

HTN planning techniques are based on tasks. The HTN planning routines in Bridge Baron 8 have a separate C function for each task that it can perform in declarer play during a bridge deal. These separate functions are very important for ease of maintenance; if Bridge Baron 8 is not performing well in particular types of situations, we can often rapidly improve its performance in many similar situations by changing the way it performs a single task, and these changes are often restricted to a single C function.

Conclusions

For games such as chess and checkers, the best computer programs are based on the use of game-tree search techniques that "think" about the game quite differently from how human players do (Biermann 1978, IBM 1997). For bridge, our new version of the 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.

Since 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, the Bridge Baron's championship performance in the *Baron Barclay World Bridge Computer Challenge* suggests that bridge may be a game in which HTN planning techniques can be very 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, 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 microwave transmit/receive modules, as part of a project that some of us have with Northrop Grumman Corporation (Hebbar et al. 1996; Smith et al. 1996b; Smith et al. 1996d; Smith 1997). Since the same approach works well in domains that are as different as these, we are optimistic that it will be useful for a wide range of practical planning problems.

Acknowledgments

This work was supported in part by an AT&T PhD scholarship to Stephen J. J. Smith, by Maryland Industrial Partnerships (MIPS) Grant 501.15, by ARPA grant DABT 63-95-C-0037, and by 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.

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 Fox, M. and Zweben, M., editors, *Intelligent Scheduling*, 451–469. Morgan Kaufmann, San Mateo, California.

Agosta, J. M. 1996. Constraining influence diagram structure by generative planning: an application to the optimization of oil spill response. *Proceedings of the 12th Conference on Uncertainty in Artificial Intelligence*, 11–19. AAAI Press, Menlo Park, California.

Biermann, A. 1978. Theoretical issues related to computer game playing programs. *Personal Computing*, Sept. 1978, 86–88.

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.

Currie, K. and Tate, A. 1985. O-Plan—control in the open planner architecture. BCS Expert Systems Conference, Cambridge University Press, UK.

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. J. J.; Minis, I.; and Nau, D. S. 1996. Plan-based evaluation of design for microwave modules. In *ASME Design for Manufacturing Conference*, p. 262 (abstract; full paper on CD-ROM).

IBM. 1997. How Deep Blue works. <http://www.chess.ibm.com/meet/html/d.3.2.html>.

Korf, R. 1994. Presentation of "Best-First Minimax Search: Othello results" at *Twelfth National Conference on Artificial Intelligence*.

Lopatin, A. 1992. Two combinatorial problems in programming bridge game. *Computer Olympiad*, unpublished.

Loy, J. 1997. Review of bridge programs for PC compatibles. Usenet newsgroup *rec.games.bridge*, 9 October 1997, Message-Id: <343CAB0B.C6E7D2B1@pop.mcn.net>.

Manley, B. 1993. Software "judges" rate bridge-playing products. *The Bulletin* (published monthly by the American Contract Bridge League), **59:11**, November 1993, 51—54.

Sacerdoti, E. D. 1974. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence* **5**:115-135.

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

Smith, S. J. J.; Nau, D. S.; and Throop, T. 1996a. A planning approach to declarer play in contract bridge. *Computational Intelligence* **12:1**, February 1996, 106–130. An earlier version is at <http://www.cs.umd.edu/TR/ UMCP-CSD:CS-TR-3513>.

Smith, S. J. J.; Nau, D. S.; Hebbar, K.; and Minis, I. 1996b. Hierarchical task-network planning for process planning for manufacturing of microwave modules. *Proceedings: Artificial Intelligence and Manufacturing Research Planning Workshop*, 189—194. AAAI Press, Menlo Park, CA.

Smith, S. J. J.; Nau, D. S.; and Throop, T. 1996c. Total-order multi-agent task-network planning for contract bridge. *AAAI-96*, 108–113.

Smith, S. J. J.; Hebbar, K.; Nau, D. S.; and Minis, I. 1996d. Integrated electrical and mechanical design and process planning. *IFIP Knowledge Intensive CAD Workshop*, CMU, 16-18 September 1996.

Smith, S. J. J.; Nau, D. S.; and Throop, T. 1996e. AI planning's strong suit. *IEEE Expert*, **11:6**, December 1996, 4–5.

Smith, S. J. J. 1997. *Task-Network Planning Using Total-Order Forward Search, and Applications to Bridge and to Microwave Module Manufacture*. Ph.D. Dissertation,

University of Maryland at College Park. <http://www.cs.umd.edu/users/sjsmith/phd>.

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

Throop, T. 1983. *Computer Bridge*. Hayden Book Company, Rochelle Park, NJ.

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*.

Wilkins, D. E. 1988. *Practical Planning.* Morgan Kaufmann, San Mateo, California.

Wilkins, D. E. and Desimone, R. V. 1994. Applying an AI planner to military operations planning. In Fox, M. and Zweben, M., editors, *Intelligent Scheduling*, 685—709. Morgan Kaufmann, San Mateo, California.