A Game Tree Strategy for Automated Negotiation

Alan H. Karp
alan.karp@hp.com

Ren Wu
ren.wu@hp.com

Kay-Yut Chen
kay-yut.chen@hp.com

Alex Zhang
alex.zhang@hp.com

Hewlett-Packard Laboratories
1501 Page Mill Road
Palo Alto, CA 94304

ABSTRACT

Chess playing programs running on small computers, such as PocketPCs, can beat most human players. This paper reports a feasibility study to determine if the techniques programs use to play chess can be applied to the more economically interesting problem of negotiation. This study allowed us to identify the essential differences between playing chess and negotiating and to demonstrate possible solutions to the problems we encountered.

Categories and Subject Descriptors

F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Computations on Discrete Structures*; H.4.2 [**Information Systems Applications**]: Types of Systems—*Decision support*; I.6.8 [**Simulation and Modeling**]: Types of Simulation—*Gaming*

General Terms

Algorithms, Economics

Keywords

Automated Negotiation

1. INTRODUCTION

Negotiation can be treated as a game, and computers have been used to play games, such as chess. However, none of the proposed strategies for automatic negotiation combines these two ideas. The strategy presented here involves treating bilateral negotiation as a two-player game and traversing the game tree that results from enforcing the protocol.

Of course, there are issues that arise in a negotiation that don't enter into playing chess. First, chess is a game of perfect information, where the payoff and game structures are common knowledge, which is obviously not true in general for bilateral negotiation. In particular, in most real world examples, the payoff is private information unknown to the opponent. Not surprisingly, the quality of the deal reached by a player depends on how good an estimate of the opponent is used. Second, in chess, if your opponent makes a bad move, you do better. That's not true in a negotiation, both parties may suffer, so a strategy must be able to deal with inept opponents. In addition, all moves in chess are discrete, the queen moves two squares or three, but some attributes in a negotiation are continuous, such as fluid measures, or effectively so, such as price.

The protocol and strategy described in this paper are not specific to the utility of the players. The only requirement for implementation is that each player can evaulate his own utility function. We also do not require a player to have knowledge of the utility function of his opponent. Instead, we have created a general model of how a player deals with uncertainties. Several experiments using additive utility functions were implemented to study the behavior of this strategy. We have found this algorithm can result in reasonable offers, in the sense that we would not be surprised to see these offers from human negotiators. These issues and others are discussed in more detail in the full report [1], which also includes a more complete list of citations. Here we'll just summarize that paper.

EC'04, May 17–20, 2004, New York, New York, USA.
ACM 1-58113-711-0/04/0005.

2. SUMMARY OF THE APPROACH

The rules controlling the negotiation process have an important influence on the viability of any strategy. For example, if negotiators exchange complete offers, they are implicitly searching a very large set of points for an acceptable deal. A good strategy is one based on some heuristic for improving the value of the deal, such as hill climbing or simulated annealing. If the protocol allows parties to change their minds, avoiding cycles becomes an important part of the strategy. One of the main design features of the protocol used for this study is the guarantee that the process will terminate in a finite number of turns.

We assume that the goal of the negotiation is to fill in the blanks in a *contract template* provided by the *marketplace* in which we're negotiating. A contract template consist of a set of *sections*. Each section defines a specific aspect of the contract, such as terms of payment or product description. The description in a section is specified in a *vocabulary*, which consists of a collection of *attributes*. Each attribute is a name-value pair and has a number of other properties, such as multiplicity and matching rule.

The negotiating parties, two in the examples studied, take turns exchanging offers. An offer consists of values, numeric ranges or sets of values, for a subset of the attributes included in the previous offer. A legal counteroffer must narrow the range of potential deals by eliminating at least one attribute value or narrowing a numeric range. Once an attribute has appeared with a single value in both an offer and counteroffer, it is *settled* and may not be reintroduced into the negotiation. A binding contract is formed when all the attributes are settled. Either party can declare the negotiation *failed* at any time before an agreement is reached. We can think of such a negotiation as a game with a finite number of positions, very much like a game of chess. Playing such a game on a computer involves building a game tree and evaluating the outcomes of each possible move.

Conceptually, building a game tree is quite simple. The root node represents the current position. Create a child node for every legal move from the node's position. If there are no legal moves from a node's position, the node is a *leaf*, and the expansion stops. Continue until there are no more moves to examine or you run out of time. We'll call leaf nodes and nodes that weren't expanded *terminal* nodes. Next, we evaluate the nodes and pick the move that has the highest payoff.

Although the basic procedure is very much like playing chess, there are a number of complications. First of all, the protocol allows compound moves, moves that could be made separately but are made in a single move. Second, we need to deal with continuous moves or moves with so many choices that we can't hope to produce a node for each possibility. One approach would be to select a modest number of values based on some heuristics as done for games with large branching factors, such as Go. Instead, we defer the problem by expanding the tree and evaluating the nodes in different steps. This choice limits, but doesn't eliminate, the opportunities to prune the tree.

Once the tree expansion is complete or we've run out of time, we compute the payoff of each terminal node. If the node is a leaf, we use the appropriate payoff function. If the terminal node is not a leaf, we use an evaluation based on all the attribute values in the offer. The value of a nonterminal node is that of the child with the largest value for the player that can move to it. Propagate these values to the root node, and the move corresponding to the child node with the largest payoff is the one to make.

Deciding how much to concede on a continuous-valued attribute at a given round is difficult. There is little information to be gleaned from game theory because of the complexity of the game; with a continuous strategic space a Nash equilibrium cannot be found even if one exists. Economic concepts do not help either, since every point is often Pareto optimal. Another field, that of making decisions under uncertainty, provides more guidance. When encountering a node corresponding to a continuous value in a decision tree, apply an algorithm, such as importance sampling, and see which value is best.

We don't have the information needed to do importance sampling, so we use a very simple optimization algorithm. Each negotiator specifies minimum and maximum concessions for each range-valued attribute. Our optimization involves computing the payoff of the node for each of these and their mean value, then fitting a parabola through these three points. The fourth try is the concession corresponding to the maximum value the parabola takes on this interval. This strategy is a simple one chosen only to illustrate the principle. A better algorithm should be used in an operational system.

Thus far, we've been assuming that we know what the other player wants. That's clearly not the case in most negotiations. There are two kinds of uncertainty. We may not know precisely the other player's payoff for a particular attribute value or how much weight the player assigns to the attribute. This uncertainty is adequately represented by a mean and standard deviation, which allows us to estimate the expected value of any offer. The second uncertainty is in the other player's constraints. I may know with a great deal of precision how much my opponent values a particular deal, but I may not be at all certain that he is able to meet its conditions.

The full paper presents several experiments that illustrate how the tree strategy operates. These results don't show any great advantage for the tree strategy because the case studied is simple enough that many strategies reach a Pareto optimal deal. More extensive testing is clearly needed, but these tests must be done in the context of a specific, realistic negotiation, which is beyond the scope of this feasibility study.

3. CONCLUSIONS

We've shown that it is possible to use the same approach to negotiation as used when programming a computer to play chess. In spite of the additional complications, the tree strategy produces results in line with expectations.

There is much more work to be done. The performance of the prototype is simply terrible, evaluating fewer than 100 nodes per second. Improving the performance will enable more systematic studies, particularly to find a better strategy for range-valued attributes. We should also study more complex contracts, and test the tree strategy against more realistic strategies, including testing it against people. We also need to quantify the cost of poor estimates of the other player's utility.

Even with dramatically better performance, we don't expect to be able to completely expand the game tree as done in our test cases; doing so will take too long and consume too much memory. Instead, we'll address the exponential explosion by intelligent pruning. The utility representation we developed allows us to calculate both the best and worst possible deals reachable from any offer. Hence, we can prune branches in which the best possible outcome is worse than the worst possible outcome on some other path. We can also use a heuristic that limits the number of changes made on any offer, since restricting the possibilities too much is likely to violate the other party's constraints. Finally, we have the advantage over chess playing programs that negotiations are often spread over days or weeks instead of minutes.

Chess playing programs have demonstrated that brute force wins over attempting to emulate human thought. We expect that a fully developed tool will demonstrate this same property for negotiation.

4. REFERENCES

[1] A. H. Karp, R. Wu, K.-Y. Chen, and A. Zhang. A game tree strategy for automated negotiation. http://www.hpl.hp.com/techreports/2003/HPL-2003-154.html.