A Game Tree Strategy for Automated Negotiation

Alan H. Karp, Ren Wu, Kay-yut Chen, Alex Zhang
Intelligent Enterprise Technologies Laboratory
HP Laboratories Palo Alto

HPL-2003-154
July 28th , 2003*

E-mail: {alan.karp, kay-yut.chen, ren.wu,alex.zhang}@hp.com

web services, automated negotiation

We present a strategy for automatic negotiation that takes the same approach as computer programs that play games such as chess; we build the game tree. For every offer we look at every counteroffer, every counteroffer to each of them, and so on. The strategy then selects the counteroffer that has the largest expected payoff.

A number of problems arise that are unique to using this strategy for negotiation. These include uncertainty in the opponent's goals, the fact that a bad move can penalize both players, and moves that are continuous, as opposed to discrete. We show how the standard methods of building the tree and evaluating the results were adapted to this environment.

A Game Tree Strategy for Automated Negotiation

Alan H. Karp, Ren Wu, Kay-yut Chen, Alex Zhang
Hewlett-Packard Laboratories
{alan.karp,kay-yut.chen,ren.wu,alex.zhang}@hp.com

July 21, 2003

Abstract

We present a strategy for automatic negotiation that takes the same approach as computer programs that play games such as chess; we build the game tree. For every offer we look at every counteroffer, every counteroffer to each of them, and so on. The strategy then selects the counteroffer that has the largest expected payoff.

A number of problems arise that are unique to using this strategy for negotiation. These include uncertainty in the opponent's goals, the fact that a bad move can penalize both players, and moves that are continuous, as opposed to discrete. We show how the standard methods of building the tree and evaluating the results were adapted to this environment.

1 Introduction

Negotiation can be treated as a game[1], and computers have been used to play games, such as chess[4]. However, none of the proposed strategies for automatic negotiation combine these two ideas. The strategy presented here involves treating the 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. For example, in chess each player knows what the other one wants; this intent must be estimated in a negotiation. Not surprisingly, the quality of the deal reached by a player depends on how good an estimate of the opponent is used. Secondly, in chess, if you make a bad move, your opponent does better. That's not true in a negotiation, both parties may suffer, so a strategy must be able to deal with inept opponents. 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.

These issues and others are discussed in this paper. Section 3 describes how the tree is built, and Section 4 explains the move evaluation process. Dealing with our uncertainty in what the other player wants is also described in Section 4. Some examples of using the tree strategy against some other, simple strategies are shown in Section 5. As with any game, you can't play unless you know the rules, which are described in Section 2.

2 Rules of Engagement

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[3]. If the protocol allows parties to change their minds, avoiding cycles becomes an important part of the strategy. This section summarizes the rules of engagement used for the tree strategy described in Sections 3 and 4. More detail is provided elsewhere [6].

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 (proper) 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.

3 Building the Tree

It is clear that the protocol described in Section 2 is guaranteed to terminate in a finite number of steps. That means that we can think of the 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, as described in Section 4 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. Secondly, 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.

3.1 Move Generator

You've got to be able to make legal moves to play a game. The same applies to negotiation; you've got to follow the protocol. In chess, the moves are quite simple, but any combination of single moves is also a legal move according to the negotiation protocol we use.

While it is possible to write a procedure to generate all possible moves, we took a different approach. We have a procedure to find every legal move that changes a single attribute value. For each, we produce a child node and a sibling. The sibling's move is a combination of the move used for the child and the move to reach the current node.

An example illustrates the procedure. Say that the offer of the current node includes two attributes,

shoe style [wingtip,loafer]
shoe color [black,brown]

We want to generate all legal moves from this position. First, we create a child to represent a simple move, say

(shoe, style, [wingtip])
(shoe, color, [black,brown])

This node produces a child node representing the move to

(shoe, style, [wingtip])
(shoe, color, [black])

and a sibling representing this offer. The sibling is equivalent to the parent node producing two children, one with the simple move that eliminated loafers and one with a compound move that eliminated both loafers and brown. Applying this approach recursively results in child nodes for all possible moves.

The advantage of this approach is the simplicity of the move generator. However, it does have a down side. We can build the tree depth first by creating the child before the sibling, but we can't generate the tree breadth first. There don't appear to be any negative implications of this limitation.

3.2 State of the World

Throughout, this work has followed the advice implicit in Knuth's statement that "premature optimization is the root of all evil" [8]. There is one optimization that was implemented, though, detecting when a position has been seen before. The primary reason this optimization is important is when we consider the game as a whole instead of just a move at a time. Building and evaluating a game tree is computationally expensive, and the number of nodes that can be evaluated is limited by the available resources. On the opening move, we have no choice but to build the tree and evaluate the nodes. However, on subsequent moves, we'd like to be able to reuse the work done previously. If the position is one we've seen before, we can simply read off the value of the node if it's a leaf or continue expanding the tree from that point if it is not.

We call a position in the tree, including the time because payoffs are generally functions of time, a *State of the World*, SoW. Clearly, comparing the attribute values one by one would be too time consuming. Instead, we use a representation similar to the Zobrist keys used in chess programs [10, 12]. Like these functions, we compute a hash function with a value that does not depend on the order in which terms are included.

The hash of the n'th term is $h(c_n)$, where c_n represents the triple (vocabulary name, attribute name, attribute value). There is one such triple for each, individual attribute value. Starting with $h(0) = 0$, compute

$$h(n) = h(n-1) + h(c_n). \tag{1}$$

This hash function can also be downdated by subtracting the hash of the term being removed. This approach is closely related to a standard hash function [9]. The only difference is that work uses a different hash function for each update because order dependence is desired. We use MD5[11] for all terms.

Using exclusive-or (XOR, denoted $\oplus$) instead of addition in Equation 1 works if the same attribute value can't appear more than once. If it could, XOR would be guaranteed to produce collisions since both $a \oplus a$ and $b \oplus b$ produce a zero. Although repeated attribute values violate the protocol, the way attributes with continuous attributes are handled does result in repeats.

At first glance, there appears to be a problem. There are exponentially many ways to form a particular sum. That observation seems to indicate that the probability that distinct deals produce the same hash value, the *collision probability*, approaches unity.

It's easy to see that the collision probability of the sum is the same as that of any single term. Say that there is some set of k numbers that produces a collision with the sum of another set. At some point, the sum will have accumulated $k - 1$ of them. What's the chance that the next number is the one that produces the collision? It's exactly $1/M$, the probability that any two inputs produce a collision in the base hash function. In fact, any set of $k - 1$ numbers is a member of a set that collides because there is always (in modulo arithmetic) a number that can be added to their sum to produce a collision. It's just rare that this particular number gets picked next.

3.3 Attributes with Continuous Values

Some attribute values can be continuous, such as liquid measures, or effectively continuous, such as price. The latter are effectively continuous because there are far too many discrete values to assign each to a node in the tree. One approach would be to select a modest number of values. The problem is knowing which values will lead to the best end result.

The approach used here is to defer the problem by expanding the tree and evaluating the nodes in different steps. As we generate the moves and instantiate the child and sibling nodes, we mark any move involving any set of one or more range-valued attributes. The exact values to be used are determined during the evaluation step. We can't terminate the tree on complete deals because we don't know how many rounds of counteroffers will occur over the continuous attributes. However, we know that a child node that represents the same state of the world as its parent is in such a situation. That fact means that the pattern of offers from the child is identical to that of the parent, and the set of children of the child is known.

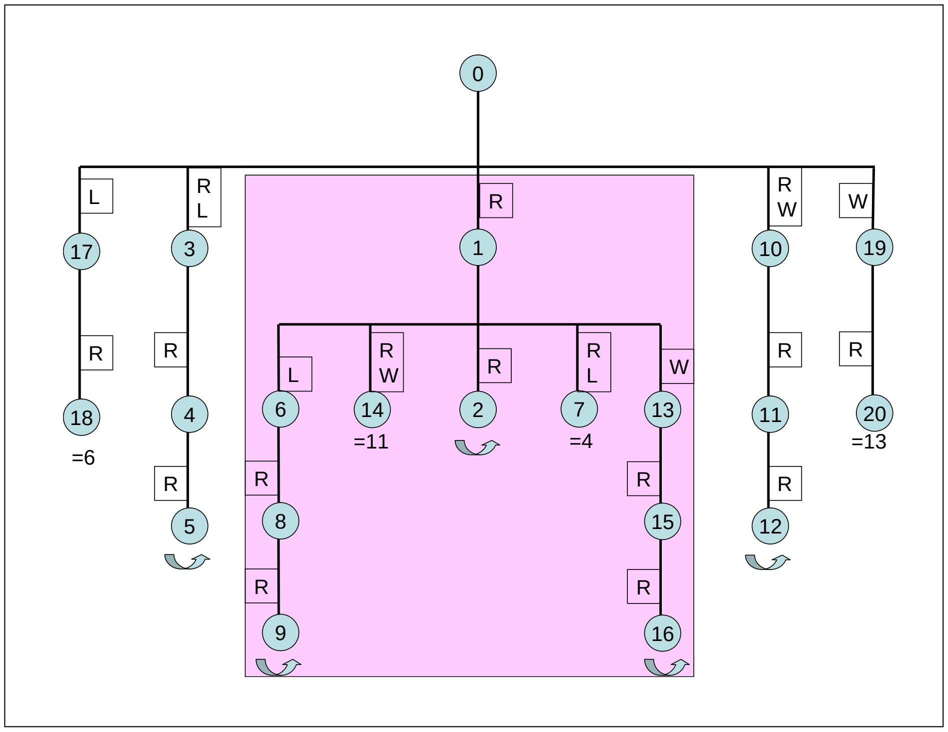

Figure 1: Tree expansion of a small example.

Figure 1 illustrates this procedure for a very simple case. The offer is for wingtips or loafers at a price between $150 and $300. We start at the root node labeled 0. Each link is labeled with a flag denoting the change made in moving to that node. For example, the state of the contract at node 17 differs from that at node 0 by having selected loafers. We use the label "R" to denote any set of changes involving range-valued attributes.

Node 1 represents offers that narrow any combination of range-valued attributes, only price in this simple example. We now expand node 1 by creating node 2, which corresponds to another counteroffer on the range-valued attributes. Node 2 next generates a sibling, node 3, which represents the move to node 2 plus another legal move, selecting loafers in this case.

Node 3 creates node 4, which corresponds to a counteroffer on only range-valued attributes. Node 3 does not produce a sibling, because the only legal counteroffer would be the same state of the world as node 3. The expansion along this branch ends when node 4 creates node 5, a node that makes the exact same change as its parent. The arrow under a node denotes that the negotiation cycles through the node and its parent until all range-valued attributes are settled. Some of the branches in the tree end early. For example, node 7 is not expanded because it represents the same state of the world as node 4.

4 Evaluating 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. For example, in chess the node will have a score of -1 if it is a win for the other player, 0 if it represents a draw, and +1 if it is a win for me. In a negotiation, we use the utility functions of the players. If the terminal node is not a leaf, use some heuristic to estimate its payoff.[1] We'll use an evaluation based on all the attribute values in the offer[6]. The value of a non-terminal 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 move corresponding to the child node with the largest value is the one to make.

Figure 2 shows the expansion of the shaded part of Figure 1 using the parameters shown in Section 5. Node 1 corresponds to a change in a range-valued attribute, price in this case, and the seller makes the minimum concession specified in the parameters, \$57. In the move to node 2, the buyer also makes the minimum concession, \$53. The subtree under node 2 is identical to that under node 1, with node 1 replacing node 2, reflecting the fact that the legal counteroffers from node 2 are identical to those from node 1. The seller's move from node 2 to the node 1 below it settles the price. The only move for the buyer to make from that point is to choose either wingtips or loafers.

We now want to compute the value of node 1 at time 1, so the seller will know if this price concession will lead to the best end result. We start at the bottom and work our way up. To avoid confusion, the buyer's payoffs are underlined. Look first at nodes 6 and 13 on the last row. If the buyer picks loafers, the payoff to the buyer is 12.5; it is 16.7 if the buyer picks wingtips. Hence, the buyer will pick wingtips for \$203. This deal has a payoff of 16.2 to the seller, so that's the value assigned to node 1 in the third row. We compute the payoffs of the other nodes the same way.

[1]The quality of these heuristics affects how strong a game a program plays.

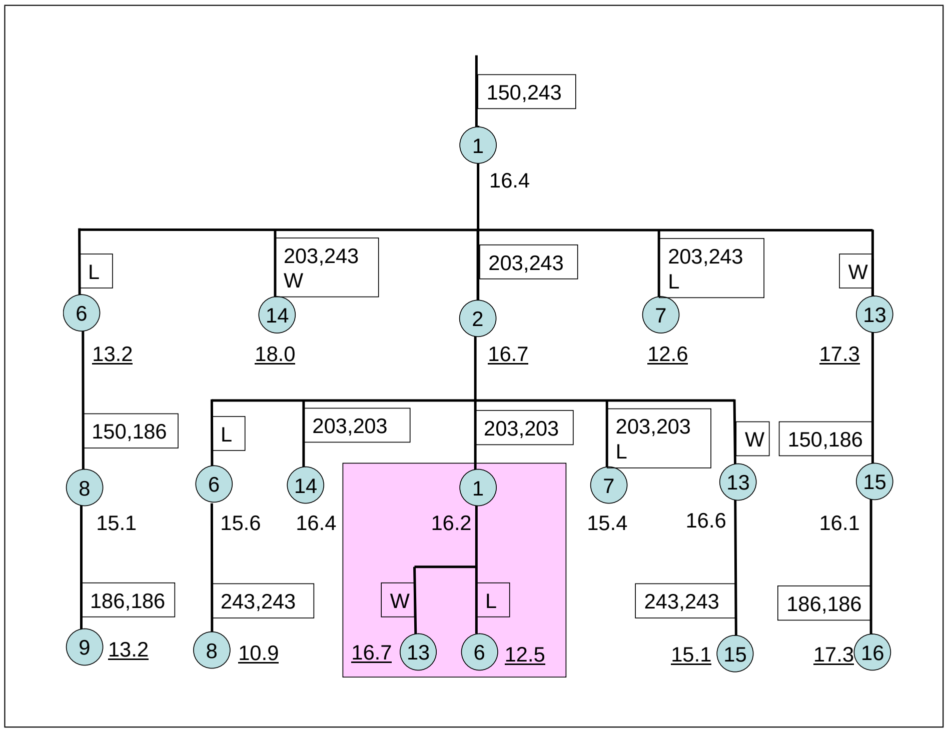

Figure 2: First evaluation of a small example.

Note that nodes 7 and 14 represent the same states of the world as nodes 4 and 11, respectively, which aren't shown in this figure.

The second row shows the payoff to the buyer. Note that node 14, corresponding to wingtips for no less than \$203, is worth more than that of node 13 in the last row. The time penalty explains the difference. The seller's move is to concede the price of \$203. Thus, the buyer gets a higher payoff by picking wingtips and conceding on price on the same turn instead of separately. Continuing up the tree, we see that the best payoff to the seller is 16.4. That's because the seller expects the buyer to make the best move, namely to select wingtips and concede on price.

We're not done, yet. There might be a different concession in the range-valued attributes that we can make going to node 1. If getting to a deal sooner has value, perhaps a larger price concession will lead to a better payoff for the seller. Figure 3 shows the result. Indeed, a deal is reached sooner, but not enough sooner to offset the difference in price, \$150 *versus* \$203.

4.1 Conceding a Continuous Attribute Value

Deciding how much to concede on a given round is difficult. There is little information to be gleaned from game theory because it usually considers only purely competitive attributes, such as price. The problem is that there is no theory to

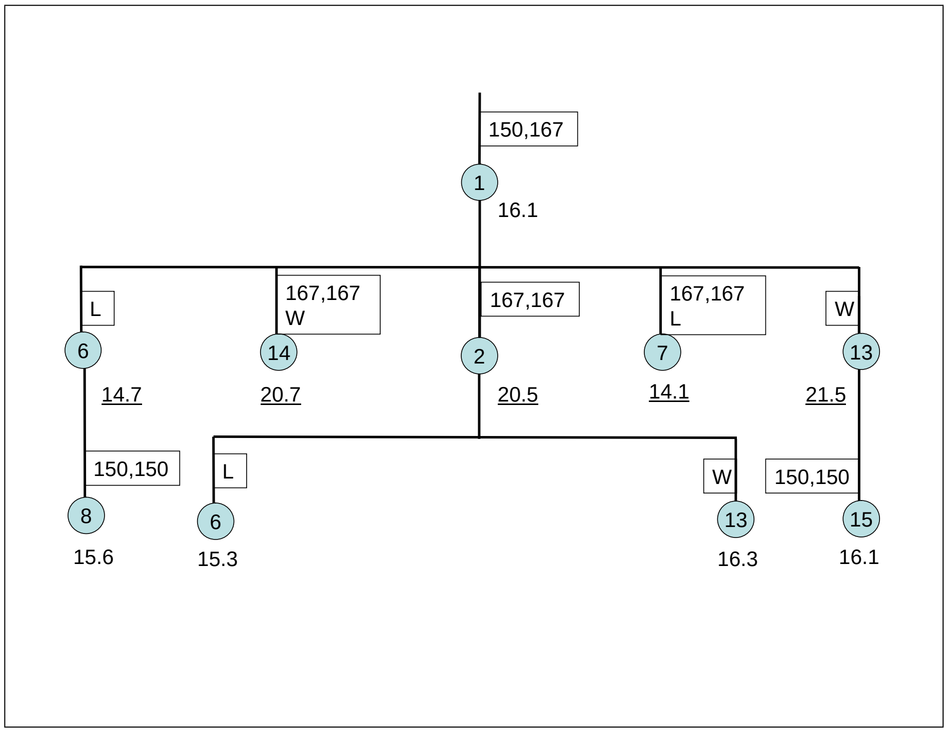

Figure 3: First evaluation of a small example.

be developed; all such points are Pareto optimal[1]. Another field, that of making decisions under uncertainty, provides more guidance[5]. 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 of the parabola if it lies on this interval. This strategy is a simple one chosen only to illustrate the principal. A better algorithm should be used in an operational system.

We'll illustrate the approach with a particularly challenging case, a linear time penalty and a linear price utility. Since the sum of two straight lines is a straight line, we know that the optimal price concession occurs for either the minimum concession or the maximum. The minimum concession is best when the price term dominates the time penalty; the maximum concession, otherwise.

The situation is more complicated in a real negotiation as shown in Figure 4. The points computed by looking at the outcome of different concessions do not lie

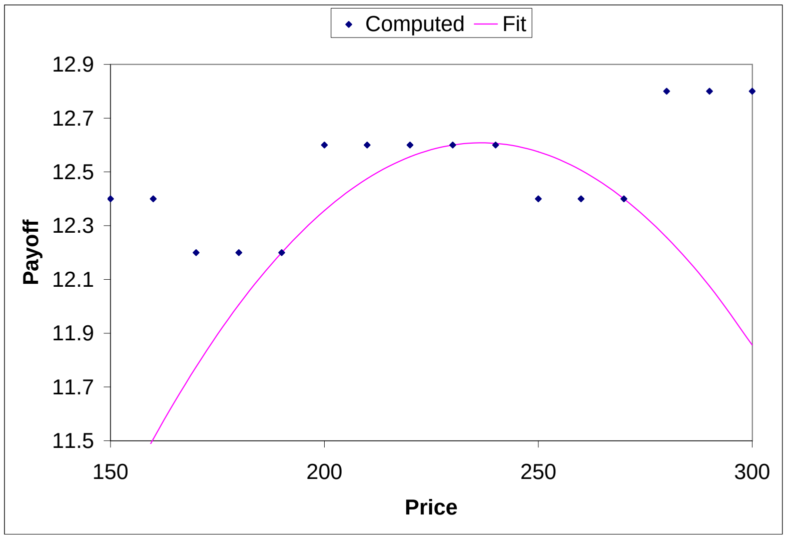

Figure 4: Best counteroffer for a range-valued attribute.

on a straight line because of the granularity of the changes. If the parties concede too slowly, the time penalty will outweigh the benefits of any deal before it can be reached, and the negotiation fails. If the parties use a larger concession, there is a parity effect. Consider an offer with everything settled except the price, which has a range of \$150 to \$170, and a time penalty such that the minimum concession is \$20. If it is the buyer's turn, the final price will be \$170; if the seller moves, the result will be \$150.

In the example shown in Figure 4, we assume that the seller's minimum and maximum concessions are \$30 and \$110, respectively. Thus, our algorithm tries the prices \$270, \$190, and \$230. The parabola through these points has a maximum at \$236.77, a concession of \$63.23, a value unlikely to have been chosen *a priori*. Running the negotiation with this concession produces a payoff of 12.6, slightly less than the payoff predicted by the fit, but better than using either of the end points. Note that a \$20 concession would have resulted in a better result due to the parity effect, but there's no way to know that without sampling a large number of points. Additionally, a concession that small is not allowed by the parameters specified by the seller.

4.2 Including Uncertainty

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. Including this factor in the evaluation step is discussed in Section 4.3.

The second uncertainty is in the other player's constraints. I may know with a great deal of precision how much a seller values black wingtips, but I may not be at all certain that they are in stock. Including this uncertainty is discussed in Section 4.4.

4.3 Payoff Uncertainty

We evaluate a node using the payoffs of its child nodes. If we know with certainty the payoff assigned to each attribute value, then we can simply take the payoff to be the largest of the payoffs of the children. However, we are assuming some mean and standard deviation for the contribution of each attribute value to the payoff.

The first problem is to compute the mean and standard deviation of a deal when all we have are means and standard deviations of the individual attribute values. In general, the distributions are not independent. For example, we might not be surprised if the uncertainty for the estimated utility of brown is correlated with that of loafers. This observation not withstanding, we assume the uncertainties are independent because we believe the uncertainty of our estimations far exceeds the error introduced by making this assumption. In this case the mean and standard deviation of the payoff of a deal can be computed from $v = \sum_i v_i$ and $\sigma^2 = \sum_i \sigma_i^2$, respectively. Here the sums are over all terms in the utility representation contributing to the payoff.

We've made the assumption that neither party will accept a deal with negative utility. Since our estimates have a spread about a mean value, we need to compute the probability that a particular deal has a positive utility for all the parties. For each player, the probability that the result has a positive payoff is

$$P_i(\overline{v}_i > 0) = \frac{1}{2}(1 + erf\left(\frac{\overline{v}_i}{\sqrt{2}\overline{\sigma}_i}\right)), \tag{2}$$

and the probability that the offer will be accepted by everyone, P_r, is the product of these probabilities. In order to properly account for this likelihood, we multiply the payoffs by these probabilities. To summarize, the adjusted payoff assigned to a node for player i is

$$v_i = \prod_{j=1}^{J} P_j \overline{v}_i. \tag{3}$$

Let's say that we have two players, Sally Seller and Bob Buyer. Sally is selecting between two nodes, 1 and 2. Call Sally's payoffs for these nodes v_{S1} and v_{S2} and

Bob's payoffs v_{B1} and v_{B2}. Sally will choose the node with the larger value, say node 1, but Bob only has an estimate of Sally's utility. In particular, Bob assigns a mean and standard deviation to his estimate of Sally's utility. Call the mean values m_1 and m_2 and standard deviations σ_1 and σ_2. We assume that these two values are uncorrelated, which may not be the case if the two child nodes are both part of the same combination, but this error can be included approximately in the standard deviations.

The probability that Sally will choose node 1 is

$$p_1 = P\{v_{S1} > v_{S2}\} = \int_{-\infty}^{\infty} \phi_1(t)\Phi_2(t)dt, \tag{4}$$

where

$$\phi_1(t) = \frac{1}{\sqrt{2\pi}} e^{(t-m_1)^2/2\sigma_1^2}, \tag{5}$$

$$\Phi_2(t) = \int_{-\infty}^{\infty} \phi_2(t)dt = \frac{1}{\sqrt{\pi}} \int_{-\infty}^{\frac{t-m_2}{\sigma_2\sqrt{2}}} e^{-x^2}dx. \tag{6}$$

Hence,

$$p_1 = \frac{1}{2\sqrt{\pi}} \int_{-\infty}^{\infty} e^{-y^2} \left[1 + \operatorname{erf}\left(\frac{\sigma_1}{\sigma_2} y - \frac{m_2 - m_1}{\sqrt{2}\sigma_2}\right)\right] dy. \tag{7}$$

This result is in a form amenable to numerical integration using Gauss-Hermite points and weights. Since erf is smooth and high accuracy isn't needed, only a few points are needed.

When the node has more than two children, we have the joint probability. So, in general, the probability that node 1 will be chosen is

$$p_1 = \int_{-\infty}^{\infty} \phi_1(t) \prod_{k=2}^{K} \Phi_k(t)dt. \tag{8}$$

Bob's expected payoff is then

$$v_B = \sum_{k=1}^{K} p_k v_{B_k} \tag{9}$$

4.4 Constraint Uncertainty

The second uncertainty is the likelihood that an attribute value or combination of values violates a constraint of one of the players. For example, I may insist on using a credit card for merchandise to be shipped to me. An offer that includes only cash or check as payment options violates my constraint. We assume such an offer has a payoff of $-\infty$ and results in a failed negotiation.

Our representation of the players' utility functions [7] includes an explicit value for the estimation that a particular combination is required, P_{ik}, for combination k in player i's utility function. The aggregate acceptance probability is

$$P_i = 1 - \prod_{k=1}^{K}(1 - P_{ik}), \tag{10}$$

which is multiplied by the probability that a deal's payoff is positive, P_r, when computing the node's payoffs for the players.

4.5 Payoff with Uncertainty

We can see how these factors enter into the node evaluations. First, consider the shaded area of Figure 2. The seller assigned a payoff of 16.2 to this node because that's the seller's payoff if the buyer picked wingtips, the buyer's best deal. However, the seller isn't sure of the buyer's valuation or constraints.

Say that the seller has a payoff of 16.2 to the deal if the buyer selects wingtips and 15.3 if the buyer opts for loafers, both with a standard deviation of 1, and a 50% probability that either wingtips or loafers violate the buyer's constraints. The seller's expected value is 16.0 ± 1.4 with an acceptance probability of 75%. Thus, when comparing this node with others, we use an effective payoff of 12.0 ± 1.1.

5 Experiments

We present 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 work.

We assume that the seller's utility function is

$$V_{seller} = -t/5 + b_w + b_l + b_{bl} + b_{br} + 3(2b_{ca} + b_{cr}) + 2(b_{[0,0]} + b_{[1,\infty]}) + H(P - 100)/100, \tag{11}$$

where t is a measure of time , $b_x = 1$ if attribute value x is in the complete deal and 0 otherwise.[2] Here w and l denote wingtips and loafers, respectively; bl and br, black and brown; ca and cr, cash and credit; $[a, b]$ delivery delays between a and b. The function $H(x) = x$ if $x \geq 0$ and is $-\infty$, otherwise. In this notation the buyer's utility is

$$\begin{aligned} V_{buyer} = \quad & -t/10 + 3[(3 - 2t/5)b_w b_{bl} + (2 - t/10)b_l b_{br}] + b_{ca} b_{[0,0]} + b_{cr} b_{[1,\infty]} + \\ & 2(b_w + b_l) + b_{bl} + b_{br} + H(300 - P)/25 \end{aligned} \tag{12}$$

Each negotiator has a set of parameters for the range-valued attributes, which are summarized below.

[2] This simplified form of the utility function can only be used for complete deals.

Table 1: Results with Random and Local Strategies

Buyer	Seller	Style	Color	Price	Time	V_{buyer}	V_{seller}
Random	Random	wingtip	brown	192.16	7	7.6	11.5
Local	Local	wingtip	black	243.00	9	3.6	11.6
Random	Local	loafer	black	250.00	9	5.1	11.7
Local	Random	loafer	black	158.39	9	8.8	10.8
Random	Tree	wingtip	black	248.09	4	9.9	12.7
Local	Tree	wingtip	black	186.00	5	11.1	11.9
Tree	Random	loafer	brown	186.00	7	11.8	11.5
Tree	Local	loafer	brown	243.00	7	9.5	12.0
Tree	Tree	loafer	brown	243.00	7	9.5	12.0

	Price		Delay	
Negotiator	Buyer	Seller	Buyer	Seller
First	50	300	3	2
Final	250	100	0	0
MinInc	53	-57	-2	-1
MaxInc	151	-153	-3	-2

A few strategies have been tested against each other [6]. Here we'll try each of them against the tree strategy. The first strategy, denoted Random, is to make a random change to an attribute value. Since the strategy doesn't check to see if the offer violates a constraint, this strategy may fail to find a deal. Several random seeds were tried to produce the results shown. The second strategy, denoted Local, examines all the attribute values and changes the one that results in the best local change without considering future changes. The third is the tree strategy described here.

All negotiations start from the same initial offer made by the buyer.

delivery delay	[0, 1]
shoe color	[black, brown]
shoe style	[wingtip, loafer]
payment price	[150.0, 300.0]
payment method	[check, cash, credit]

The strategies produce the deals summarized in Table 1. The first two columns are the buyer and seller strategies, respectively. The next three columns are the agreed upon style, color, and price. All combinations ended with a deal for cash and a delay of 0. The last two columns are the payoffs to the participants. We can't say that the seller got a better deal because these payoffs are independent; they may even be in different units. What we can compare are the payoffs between deals.

All these runs were made with both negotiators making the minimum concession for range-valued attributes. Doing so makes comparisons among the strategies simpler by removing one variable from the analysis. Secondly, the tree strategy would always pick the minimum concession in price in this case because of the balance between the time penalty and the price dependence. Since each price tried in the tree strategy increases the run time by a multiplicative factor, trying only one price saved a considerable amount of computer time.

The most notable feature of these results is the poor behavior for the buyer when both use the Local strategy. The reason is the time dependence of the buyer's utility for black wingtips and brown loafers. In the negotiation, the buyer selects black at time 2, a time when black wingtips have a higher payoff than brown loafers. However, a deal isn't reached until time 8, a time when brown loafers have a much higher payoff. That doesn't happen when the buyer uses the tree strategy.

6 Summary

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.

One important issue that hasn't been addressed in the current work is the question of what happens when the tree doesn't fit in memory. It is not clear whether the approach taken here should require more or less memory than the standard approach. On the one hand, the full tree will be smaller because all moves involving range-valued attributes are lumped into a single node. On the other hand, we can't prune the tree while it is being built because we defer the evaluations. The working set in memory will probably be smaller, because we visit the same data structures many times during the evaluation phase, as shown in Figure 3. Clearly, this issue needs more study.

References

[1] K. Binmore, *Fun and Games: A Text on Game Theory*, D. C. Heath and Company, Lexington, MA, 1992

[2] *Multinomial Probit: The Theory and its Application to Forecasting*, Academic Press, NY, 1979

[3] P. Faratin, M. Klein, H. Sayama, and Y. Bar-Yam, "Simple Negotiating Agents in Complex Games: Emergent Equilibria and Dominant Strategies", in *Proceedings of the 8th Int Workshop on Agent Theories, Architectures and Languages (ATAL-01)*, Seattle, USA, pp. 42–53, 2001, also at http://ccs.mit.edu/peyman/pubs/ATAL-01.pdf

[4] F.-H. Hsu, *Behind Deep Blue*, Princeton Univ. Press, 2002

[5] G. Infanger, *Planning Under Uncertainty*, South-Western Publishing, Danvers, Mass., 1994

[6] A. H. Karp, "Rules of Engagement for Web Service Negotiation", HP Labs Technical Report HPL-2003-152, 2003 http://www.hpl.hp.com/techreports/2003/HPL-2003-152.html

[7] A. H. Karp, "Representing Utility for Automated Negotiation", HP Labs Tech Report HPL-2003-153, 2003, http://www.hpl.hp.com/techreports/2003/HPL-2003-153.html

[8] D. E. Knuth, *Literate Programming*, Center for the Study of Language and Information, Stanford, CA, (CSLI Lecture Notes, no. 27.) 1992

[9] D. Knuth, "The Art of Computer Programming: Sorting and Searching", pg. 519, Addison-Wesley, 1998

[10] B. Moreland, "Zobrist Keys", http://www.seanet.com/ brucemo/topics/zobrist.htm

[11] R. Rivest, "The MD5 Message Digest Algorithm", RFC 1321, http://www.ietf.org/rfc/rfc1321.txt, 1992

[12] Zobrist, A. L., "A Hashing Method with Applications for Game Playing", Technical Report 88, Computer Science Department, University of Wisconsin Madison 1970, reprinted in *International Computer Chess Association Journal*, **13**, #2, pp. 69–73, 1990