Representing Utility for Automated Negotiation

Alan H. Karp
Intelligent Enterprise Technologies Laboratory
HP Laboratories Palo Alto
HPL-2003-153
July 28th , 2003*

E-mail: alan.karp@hp.com

utility functions, automated negotiation

We present a means to represent utility, the measure of goodness of a possible deal. This representation includes a number of features necessary to represent complex requirements, such as time dependence, explicit combinations of terms, and cross dependences. The formulation is closely tied to the form used to represent contracts, which makes it useful for automated negotiation software.

* Internal Accession Date Only
Approved for External Publication

Representing Utility for Automated Negotiation

Alan H. Karp
Hewlett-Packard Laboratories
alan.karp@hp.com

July 21, 2003

Abstract

We present a means to represent utility, the measure of goodness of a possible deal. This representation includes a number of features necessary to represent complex requirements, such as time dependence, explicit combinations of terms, and cross dependences. The formulation is closely tied to the form used to represent contracts, which makes it useful for automated negotiation software.

1 Introduction

Implicit in the concept of negotiation is an evaluation function that assigns payoffs to different deals; the most common representation being a multi-attribute utility function [6]. In effect, the utility function is a mapping from the many attribute values in a potential deal to a single number, which simplifies comparing deals.

Determining your own utility function is not always easy [7], and estimating that of the other party in the negotiation is even harder. An alternative approach is to rank order the possible deals. Software tools have been developed to make this problem tractable [3].

Much of the work on multi-attribute utility theory (MAUT) involves the mathematical properties or the construction of utility functions from complex and conflicting requirements [6, 7]. The actual specification of the various terms is beyond the scope of that work, so the examples shown are *ad hoc* [1]. We need something more precise to use utility functions in automated negotiation systems. In this paper, we'll use the formal specification of the terms in a contract as the basis for representing the utility function. In addition, we'll show how to use this representation to express dependences among various attribute values.

We'll start by describing the respresentation as it appears in the utility theory literature in Section 2. Section 3 extends this approach to allow us to compute the best deal possible from a given point in the negotiation based on a offers that can include multiple values for an attribute [4]. We next extend the representation to account for explicit dependences between attributes in Section 4. We argue in

Section 5 that price equivalence is not always an appropriate way to value deals, and discuss in Section 6 the use of partial order instead of utility.

Special attention is paid to constraints on attributes with continuous values in Section 7. The impact of time is described in Section 8. Section 9 contains the formal specification we used in our prototype system [4]. In a negotiation, even if we know what we want, we may only have an estimate of what the other party wants; Section 10 and 11 show how this uncertainty can be included in our utility representation.

2 Additive Utility Function

Let $j \in \{1 \ldots J\}$ be the attributes being negotiated, and $\vec{x}_j$ be the K_j-element vector of values for the j'th attribute.[1] A *deal* consists of a single value for each of the J attributes that is included in the final contract.

Each of the negotiating parties, i has a value, $V_i(\vec{x}_j)$ representing its payoff. This general formulation is often replaced by a linear combination of terms. We use the somewhat more general form shown in Equation 1 to clarify the extension to compound offers in Section 3.

$$V_i = \sum_{j=1}^{J} w_j^i \sum_{k=1}^{K} b_{jk} V^i(x_{jk}), \tag{1}$$

where $V^i(x_{jk})$ is the contribution to the payoff negotiator i assigns to the k'th value of attribute j, and w_j^i is a weight function that adjusts the scaling of these functions to reflect the relative importance of the j'th attribute to negotiator i. This *additive utility function* implies *mutual utility independence* but is more restrictive [6].

The term b_{jk}, the k'th element of the vector $\vec{b}_j$, is 1 if the corresponding attribute value is included in the offer, and is 0 otherwise. Only one of the elements of $\vec{b}_j$ is 1 for this form of the utility function, but all elements may be zero if the attribute is not included in the offer. The k'th element of the vector $V^i(\vec{x}_j)$ has the contribution of the corresponding attribute value to the payoff. Hence, the summation over k is an inner product of these two vectors. The result is the contribution to the payoff of the j'th attribute if the k'th value appears in the offer.

3 Evaluating Compound Offers

An offer in the protocol we use may include multiple values for an attribute [4]. We can't use Equation 1 to determine the payoff, because it sums the payoff contribution of all the attribute values in the offer. It is unlikely to be the case that a potential deal is worth more if both black and brown are still options, but that is exactly what Equation 1 implies when more than one value for a given attribute appears in the offer.

[1] This list is replaced by a range for numeric attributes [2]. However, we can treat the range as a finite vector of discrete values or an infinitely long vector representing a continuum of values.

In order to properly represent the utility, we modify the definition of the representation. First of all, we note that we'd like the utility to be $-\infty$ for attribute values that violate our constraints, and those attributes that are not included in the offer should not affect the utility. Constraints are easily represented in this form using a penalty function; $V^i(x_{jk})$ is set to $-\infty$ if the attribute value violates a constraint. We also change the two values that appear in the binary vector $\vec{b}$ to be 0 if the attribute value is in the offer and $-\infty$ if it is not.

The result of adding these two terms is the contribution to the payoff, with a result of $-\infty$ if the value violates the constraints when the attribute is included in the offer, and $-\infty$ if the attribute value is not in the offer. The modified form of the utility function becomes

$$V_i = \sum_{j=1}^{J} w_j^i \max_{k=1}^{K_j} [b_{jk} + V^i(x_{jk})], \tag{2}$$

which represents the best deal that can be reached from this offer. This form still requires additive independence, with all the assumptions that condition implies.

It is not necessary for an attribute to be introduced into the negotiation at all [4]. However, unless every attribute appears, we can't simply take $b_{jk} = -\infty$ for all values of this attribute. Doing so would make the utility $-\infty$, indicating that no agreement is possible. We deal with this fact by assuming there is a payoff of zero for each attribute representing its absence in the offer. Element $K_j + 1$ of $\vec{b}_j$ is normally zero if the attribute is not included in the offer and $-\infty$ if it is. The only exception is that this element will be $-\infty$ if the attribute is required by the negotiating party. We only need to choose no deal over one with a zero utility to make this formulation work in all cases.[2]

4 Composite Utility Function

While it is simple, the formulation of Section 3 misses the connection between attributes. For example, I can express a preference for black wingtips over brown loafers, but no assignment of values and weights can capture the fact that I also prefer brown loafers to brown wingtips and black loafers.[3]

The proof is simple. Say that w_1 represents the weight factor for style and w_2 the weight for color. The two styles will be denoted by x_1 and x_2 while the two colors by y_1 and y_2. Our conditions are

$$w_1 x_1 + w_2 y_1 > w_1 x_2 + w_2 y_2 \tag{3}$$

$$w_1 x_2 + w_2 y_2 > w_1 x_1 + w_2 y_2 \tag{4}$$

[2]We can't simply take this term to be zero, because the max function would select it instead of a negative value for one of the included attribute values. If there are other terms that make the utility positive, selecting zero instead of a negative value for the contribution of an attribute will lead to an overestimate of the utility.

[3]Note the absence of a comparison between brown wingtips and black loafers. These two deals can't be compared, resulting in a partial order of the deals.

$$w_1 x_2 + w_2 y_2 > w_1 x_2 + w_2 y_1 \tag{5}$$

Equation 4 implies that $w_1(x_1 - x_2) > 0$, and Equation 5, $w_2(y_2 - y_1) > 0$. Thus, $w_1(x_2 - x_1) + w_2(y_2 - y_1) > 0$, which contradicts the Equation 3.

This weakness is a direct result of assuming additive independence. A standard way to deal with this problem is to introduce a pseudo attribute [6]. Rather than use one of the standard approaches, we adopt a utility function of the form of Equation 2 for each enumerated set of attribute values. In other words,

$$V_i = \sum_{j=1}^{J} w_j \max_{k \in C_j} [b_{jk} + V^i(x_{jk})] + \sum_{j=1}^{J} w_j \max_{k=1}^{K_j} [b_{jk} + V^i(x_{jk})], \tag{6}$$

where the first term is over the various combinations and the second term is over the remaining attribute values. The set C_j represents the set of attribute values appearing in combination j. For example, a component of this set might stand for "color=black" and "style=wingtips". In essence, we've introduced a new attribute, style-color, with the cross product of the values appearing in the offer. Note that we must use combinations to represent something as simple as a different price for wingtips and loafers. Also, the coefficients must be adjusted to avoid double counting when an attribute value appears both singly and in a combination.

We need to be careful. If one of the relevant attribute values or combinations doesn't appear in the offer, the corresponding element in V_i will be $-\infty$. If none of the terms in the summations satisfies the constraints, then the term will contribute $-\infty$, and it will appear that no deal is possible. For example, if the only element in the first summation is red loafers, which I do not want, it will appear that no deal is possible, even though black and brown may be in the offer. The solution is to make sure that each of the terms in Equation 2 includes an entry with zero value if none of the combinations is included in the offer. If we will only accept one of these combinations, then we don't include this extra component.

At first glance, it appears that there is no need for the summation over j in the first term of Equation 6 because we're enumerating the combinations. However, we need this factor when we introduce more than one pseudo attribute. For example, I prefer black wingtips to brown loafers, but I also prefer to pay cash for immediate delivery and by credit card if the merchandise is shipped. The weight factor in the first term allows me to express the relative importance of these combinations and reduces the number of terms in the summation compared to enumerating all possible combinations of the four attributes.

The cost of this approach, of course, is that the number of combinations is the product of the number of values. In the worst case, we can include every attribute value in one pseudo attribute. However, we expect that most of the time, the number of attribute values in any combination will be modest. For example, we might have one combination for style and color and another for shipping and payment methods. Our representation will be far more compact than trying to represent all possible combinations of all four of these attributes, yet it is rich enough to represent our intent. Other attributes can be included in the second term.

Note that the combinations are over attribute values, not attributes, greatly reducing the number of elements in the first term. For example, shoes might come

in 5 colors and 7 styles, but we are only interested in the colors black and brown and the styles wingtips and loafers. This example has only 4 elements in the first term, instead of 35 if we considered every possible combination of attribute values. The second summation has the remaining 3 terms for color and 5 terms for style.

It's easy to see how we could be misled by using an additive utility function that combines the preferred deals. Say that the utility function includes an element representing delivery schedule. I may not care if black wingtips are delivered in 1 day or 3 days , but I do care for brown loafers. A linear combination of V^1_{buyer} and V^2_{buyer} that properly computes the utility will necessarily have a larger value for delivery in 1 day as opposed to 2. This affect is handled by combining delivery with style and color into a composite term. Thus, there is one term for black wingtips, another for brown loafers delivered in 1 day and another for brown loafers delivered in 3 days.

We've said nothing about brown wingtips and black loafers, so we might assume they don't satisfy our constraints. Alternatively, we can continue this process to include brown wingtips as a separate element in the vector, making sure their utility is less than the above cases. However, a simpler approach is to include them as terms in the sum over individual attribute values,

$$\begin{aligned} V_{buyer}(wingtips, brown) &= 2w_1 + w_2 \\ V_{buyer}(loafers, black) &= 3w_1 + w_2, \end{aligned} \tag{7}$$

Providing such terms in the utility function is equivalent to stating that we would purchase one of these configurations given the right circumstances.

5 Price Equivalence

One problem with the linearized form of the utility function, Equation 1, is that the utility function reduces every option to a monetary value whenever price appears as an attribute. More generality is needed. Not every attribute value can be associated with a price, even if we allow negative prices [6]. For example, I may be willing to pay with cash if I take the shoes with me, but I insist on using a credit card if the shoes are to be delivered. However, we've written an equation with one term for price and another for delivery and payment options. If we're not careful, the equations will predict that there is a price at which I'll pay cash for later delivery. Indeed, there is such a price, just not a reasonable one.

This work is based on the presumption that not every price offer is reasonable, even allowing for negative prices. The objective is to avoid either party taking an unneeded object. We assume there is a smallest positive price that an item will carry, the price the seller thinks someone else will pay. This price may be only the scrap value, but it won't be zero. We can even allow a negative price as long as it doesn't exceed the disposal costs. Economists would say that we do not have *free disposal*. In other words, there is a cost associated with getting rid of junk. Another effect of this assumption is that we assume there is no reasonable price at which I'll take my second choice as long as an offer that doesn't violate my constraints for my first choice is on the table. The weight factors explicitly included in the

utility functions presented here allow me to guarantee this behavior by making the coefficient of the price term small enough that the utility of my second choice will not exceed that of my first choice for any reasonable price.

6 Utility as a Partial Order

In a negotiation, I only need to know that I prefer one possible deal to another; I don't need to know by how much more. Hence, utility need only be specified as a partial order of the possible contracts [6]. I prefer black wingtips to brown loafers, but there is no need to quantify how much. If I can get black wingtips at a price I can afford, I won't consider brown loafers at any (reasonable) price.

While Equation 2 avoids combinatoric explosion, it has a problem for the representation of offers of the protocol of our automated negotiation prototype [4]. Namely, Equation 6 does not address the issue of rank ordering the deals. However, a simple modification allows us to solve this problem. We write

$$\vec{V_i} = \vec{b_r} + \vec{V^i}(\vec{x_r}) + \sum_{j=1}^{J} w_j \max_{k \in C_j}[b_{jk} + V^i(x_{jk})] + \sum_{j=1}^{J} w_j \max_{k=1}^{K_j}[b_{jk} + V^i(x_{jk})], \quad (8)$$

where $r \in R$, and the set R represents those attribute values and combinations relevant to the rank ordering. The utility is now a vector with one component for each rank ordered deal. The first element is the most preferred deal, and so on. The summations contribute a constant value to the vector and don't affect the rank ordering.

In our example, the attribute values that affect the rank ordering are the colors black and brown and the styles wingtips and loafers. Other attributes affect the evaluation of the deal and its constraints, but are not relevant to the rank ordering. For example, I want to pay less, but I still want black wingtips more than brown loafers.

Unfortunately, not every strategy can use the form of Equation 8. Some strategies assign a probability of reaching any potential deal. In order to select a good counteroffer, they must weigh the benefit against the likelihood of reaching a particular deal. For example, if the best deal I could get is unlikely to be acceptable to the other party, I'd be better off driving the negotiation in the direction of my second best deal rather than risking a failed negotiation. Knowing when to make this tradeoff requires quantifying the amount by which one deal is favored over another. On the other hand, constraint relaxation strategies, in which I counteroffer with my next most favorable deal, can treat utility as a partial order.

7 Attributes with Continuous Values

We said in Section 2 that attributes with continuous values, such as price, can be treated as an infinite list of individual values. That's fine for the formal representation, but not when it comes time to evaluate the function.

Functions of continuous variables and those attributes with a large number of discrete values, such as price, need to be represented in functional form. We will represent the constraints by giving the function limited support and define the value outside that support to be $-\infty$. We'll use the notation $H(a, f(x), b)$, where a and b represent the lower and upper bounds of the support. For example, the seller in our shoe shopping example would use $H(100, P, \infty)$ to represent the way utility changes with price. The buyer uses $H(0, 200 - P, \infty)$ to represent the contribution of price to utility.

There is no reason the function must be linear. For example, I might value the amount of gasoline as $H(5, Pe^{-x}, 200)$, which states that it's not worth my while to buy less than 5 gallons, that I can't store more than 200 gallons, and that the last gallon is worth less to me than the first. In addition, the function H can be replaced with a representation with continuous derivatives. For example, $\tanh(\alpha x)$ is +1 for large, positive values of x, and -1 for negative values of large magnitude. Picking a sufficiently large value for α makes this function as steep as desired.

8 Accounting for Time

We also need to include the "monetary value of time", which includes more than just the time value of money [6]. Time is not just money. It has several effects on the evaluation of an offer. Sometimes time appears as a cost of negotiating, such as the resources consumed in each round or the cost to remain connected. In this case, we can simply subtract a linear term in the utility function. Other times, time enters non-linearly, as when there is a deadline. In this case, we subtract a nonlinear term, which may be different for each deal. We may also change the relative payoffs of various attribute values or their combinations, so we make $V_i(x_{sk}, t)$ functions of time, where s denotes either a combination of attribute values or a specific attribute.

There is also an effect if we choose to rank order potential deals. In this case, we make the values $V_i(x_{rk}, t)$, where r denotes one of the rank ordered deals, functions of time. If time is discrete, then these functions can be chosen to change the rank order of various possible deals without introducing overlap. This guarantee can also be made with continuous time if the function is discontinuous in time. However, it may also be that we become more accommodating of offers other than the highest ranked as time goes on. In this case, the values of the time dependence can be adjusted so that the values of the utility functions are made equal. In this case, we have, in effect, a single utility function representing the combination of deals.

9 Representing Utility

We now have a formulation for calculating utility, but we still need a way for negotiators to express their utility for evaluation. Such a representation must capture three key components, combinations, individual attributes, and time. Time in particular, but other components as well, must be representable as arbitrary functions.

One representation is a 2-element list. The first element describes the time

dependence of the cost of the negotiation. The second is a list of contributions to the payoff.[4] Each entry in the second term is a 3-element list. The first component of the combination list is the weight given the combination; the second, a list consisting of the specific contributions to the combination; and the third, a function representing the value.

A key part of this representation is an unambiguous means of identifying the attribute. Since these utility functions are to be used in the framework of negotiating contracts, we'll use the same ontology representation as used to specify the contract template [4]. Each section of the contract is expressed in a *vocabulary*, which consists of attribute-value pairs. Each section has a name unique to the contract it is in; each section is described in a set of vocabularies; each attribute has a name unique to the vocabulary it is in. Hence, we can uniquely identify an attribute by giving its section, vocabulary, and attribute names. The contract being used is implicit in the negotiation.

In more formal terms we have

payoff:: (timedep, combinations)
combinations:: combination | (combinations)
combination:: (weight, attributes, values)
weight:: Integer
attributes:: attribute | (attributes)
attribute:: (sectionName, vocabName, attName, values)
sectionName:: string
vocabName:: string
attName:: string
timeDep:: value
values:: [value] | [values]
value:: string interpreted as an executable expression | constant
constant:: string | integer | float

An example will illustrate the features of this representation. We'll assume that the shoes are described in Section 1, the payment terms in Section 2, and the delivery information in Section 3. The section and vocabulary information are redundant in this example because each section uses only a single vocabulary. Other cases may arise. For example, we may want to have separate payment terms for the product and the shipping costs. (The section name has been elided to save space.)

```
(
'-0.1*time',
(3,(
((('1','shoe','color',['black']),('1','shoe','style',['wingtip'])), '3-0.4*time'),
((('1','shoe','color',['brown']),('1','shoe','style',['loafer'])), '2-0.1*time')))
(1,(
((('3','delivery','delay',[0,0]),('2','payment','method',['cash'])),1),
((('3','delivery','delay',[1,BIG]),('2','payment','method',['credit'])),1)))
```

[4] If deals are rank ordered, there is a third term, which contains a list of the individual terms.

```
(2,(
((('1','shoe','style', ['wingtip']),),1),
((('1','shoe','style', ['loafer']),),1)))
(1,(
((('1','shoe','color', ['black']),),1),
(((('1',shoe','color', ['brown']),),1))))
(0.04,(
((('2','payment','price',[-INFINITY,200.0]),),"300.0-min(att.value)")))
)
```

The first term shown is the dependence on time. Here it is a simple linear penalty, but it could be any executable expression. We see in this example a weight of 3 given to the combination of color and style, while a weight of 1 is assigned to the combination of delivery delay and payment method. You'll see the use of the vocabulary name, attribute name, and list of attribute values to specify the exact attribute value being evaluated. Note the functional dependence on time of the color/style combinations. Also note the use of the value of the designated attribute in the last line of the example.

Although brown loafers and black wingtips appear as a combination, these four attribute values also appear as individual terms. Thus, brown wingtips and black loafers represent valid deals. On the other hand, the delay and payment method attributes appear only in particular combinations. Any other combination results in no deal.

10 Estimating Payoffs

When negotiating, we may know what we want, but we don't know what the other party wants; we must make an estimate. The better the estimate, the stronger our negotiating position will be.

In order to code our estimates for whatever strategy is being used, we modify the representation of Section 9. Each payoff value is augmented with the standard deviation of the estimate of the other party's utility. In the following example, changes from the representation without uncertainty are underlined.

```
(
('-0.1*time', '0.02*time'),
(3,(
((('1','shoe','color',['black']),('1','shoe','style',['wingtip'])),('3-0.4*time',0.5)),
((('1','shoe','color',['brown']),('1','shoe','style',['loafer'])),('2-0.1*time',0.5))))
(1,(
((('3','delivery','delay',[0,0]),('2','payment','method',['cash'])),(1,0.2)),
((('3','delivery','delay',[1,100]),('2','payment','method',['credit'])),(1,0.2))))
(2,(
((('1','shoe','style', ['wingtip']),),(1,0.1)),
((('1','shoe','style', ['loafer']),),(1,0.2))))
(1,(
```

```
((('1','shoe','color', ['black']),),(1,0.5)),
((((shoe','color', ['brown']),),(1,0.5)))))
(1,(
(('3','delivery','delay',[0,5])),(1,0.2))))
(0.04,(
(('2','payment','price',[-INFINITY,200.0]),),("300.0-min(att.value)",50.0))))
)
```

An interesting aspect of this representation is that we can also express uncertainty in our own utility function. How to interpret that uncertainty is left as an exercise for the reader. However, it might be interesting to simulate a negotiation both with and without that uncertainty to see what happens.

11 Constraint Uncertainty

I may have a very precise estimate of how much you value a particular item, perhaps something easily exchangable such as a barrel of oil, but I may not know if you have one to sell. Hence, we need to represent explicitly our uncertainty in the other party's constraints, which we do by adding a term following the weight associated with each combination. As before, additions to the previous example are underlined.

```
(
('-0.1*time', '0.02*time'),
(3,0.3,(
(('1','shoe','color',['black']),('1','shoe','style',['wingtip'])),
('3-0.4*time',0.5)),
(('1','shoe','color',['brown']),('1','shoe','style',['loafer'])),
('2-0.1*time',0.5))))
(1,0.8,(
(('3','delivery','delay',[0,0]),('2','payment','method',['cash'])),(1,0.2)),
(('3','delivery','delay',[1,100]),('payment','method',['credit'])),(1,0.2))))
(2,0.9(
(('1','shoe','style', ['wingtip']),),(1,0.1)),
(('1','shoe','style', ['loafer']),),(1,0.2))))
(1,0.8(
(('1','shoe','color', ['black']),),(1,0.5)),
((((shoe','color', ['brown']),),(1,0.5)))))
(1,0.3,(
(('delivery','delay',[0,5])),(1,0.2))))
(0.04,1.0(
(('payment','price',[-INFINITY,200.0]),),("300.0-min(att.value)",50.0))))
)
```

In this example, we estimate that there's a 30% chance that you will only buy black wingtips or brown loafers, and that you are certain to walk away from a deal that violates your price constraint.

12 Summary

In this report, we've extended the simplest form of the utility function to include cases that violate its assumption of additive utility independence and to represent uncertainty in both values and constraints. We've also shown how to express the utility in a language based on the ontologies used to represent offers.

There are a number of possible extensions. We've used the additive form for the utility function, but the multiplicative form is more general. We don't know if we need to extend the formulation, or if the pseudo attributes we introduced in Section 4 make the additive formulation sufficiently expressive. We also need to investigate the effect the utility independence assumptions have on the expressiveness of this representation. For example, we can include the value of the attribute designated in a term, but there is no way to include the values of other attributes. It is not clear if this limitation significantly restricts our ability to express our desires.

Acknowledgements: I'd like to thank Claudio Bartolini for his careful reading of a draft of this report and for the improvements he suggested.

References

[1] M. Barbuceanu and W-K Lo, "Multi-attribute Utilty Theoretic Negotiation for Electronic Commerce", in Agent-Mediated Electronic Commerce III, Current Issues in Agent-Based Electronic Commerce Systems, Frank Dignum, Ulises Corts (Eds.), also Lecture Notes in Computer Science 2003, ISBN 3-540-41749-4, Springer (2001)

[2] 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

[3] V. S. Iyengar, J. Lee, and M. Campbell, "Q-Eval: Evaluating Multiple Attribute Items Using Queries", EC'01, October 14–17, Tampa, FL (2001)

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

[5] A. H. Karp, R. Wu, K-Y. Chen, and A. Zhang, "A Game Tree Strategy for Automated Negotiation", HP Labs Tech Report HPL-2003-154, 2003, http://www.hpl.hp.com/techreports/2003/HPL-2003-154.html

[6] R. L. Keeney and H. Raiffa, *Decisions with Multiple Objectives: Preferences and Value Tradeoffs*, Cambridge University Press (1993)

[7] H. Raiffa, *The Art and Science of Negotiation*, Harvard University Press, 1982