Getting Agents to Negotiate Good Deals: A Progress Report

Alan H. Karp
Software Technologies Laboratory
HP Laboratories Palo Alto
HPL-2002-161
June 3rd , 2002*

negotiation, bargaining, negotiation strategies

One aspect of web services is automating processes formerly done by people. One such process is negotiation. Automating negotiations requires that we be able to specify what we want, and the software needs a strategy for achieving a good result. This work focuses on developing a negotiating strategy, but problems in description, defining utility, and estimating the other party's goals are also addressed. Although much work remains to be done, this report describes the progress made to date.

* Internal Accession Date Only
Approved for External Publication

Getting Agents to Negotiate Good Deals: A Progress Report

Alan H. Karp
HP Labs

Abstract

One aspect of web services is automating processes formerly done by people. One such process is negotiation. Automating negotiations requires that we be able to specify what we want, and the software needs a strategy for achieving a good result. This work focuses on developing a negotiating strategy, but problems in description, defining utility, and estimating the other party's goals are also addressed. Although much work remains to be done, this report describes the progress made to date.

1 Introduction

As more business moves to the Internet, they are coming to rely more and more on software to do business on their behalf. Although the word *agent* has been widely abused, I'll take the risk of using it for this software. Among the things this software must do are:

1. Find the service.
2. Negotiate terms and conditions, and form a contract.
3. Find the service's interface and protocol if not already known.
4. Invoke the service.

You'll notice that negotiation and contract formation appear as a single step. That's because negotiation without contract formation is just playing games. Combining negotiation and contract formation may make extending the protocol to many parties easier.

The word *negotiation* often causes confusion because it is used to denote very different activities, auctioning and bargaining. Auctioning, either direct (buyers bid) or reverse (sellers bid), is the simpler form of negotiation, but it naturally includes multiple parties. Bargaining, involving multiple rounds of offers and counteroffers, is more complex, and the extension to multiple parties is less obvious. This work uses the word negotiation in the sense of bargaining only.

This report focuses on item 2 in the above list, negotiation and contract formation. Before starting the discussion, though, it's important to know how we describe what we're negotiating for. Section 2 describes the method used in this work. As used in e-commerce, the term *contract* has the special meaning described in Section 3. In order to make progress, I'm doing this work in the framework of the negotiation protocol described in Section 4. An important part of negotiating good deals is knowing how good a deal is. Section 5 describes how we decide the best deal that we might obtain from a given point in the negotiation. I'll use that evaluation in Section 6 to develop a strategy that takes the negotiation in a direction that gives us the best deal. An example is

worked through by hand in Section 7. In Section 8, I'll describe an experimental system used to measure the quality of the algorithm.

This document is a progress report, not a formal publication. Hence, it doesn't have the expected section on related work. Some references are given as a guide to the literature.

2 Vocabularies and their Attributes

Key to the negotiation protocol and mechanisms described in this report is the concept of an *ontology* that provides the semantic content of the terms in the negotiation. Basically, an ontology provides a context for describing something. For example, the keyword "size" may be used to advertise an item, but is it my shirt size or my shoe size? If the term appears in an ontology related to shoes, we know. Of course, true semantic content depends on human understanding. After all, the word "size" has meaning only to someone who speaks English.

There are many ways to represent ontologies; here I'll use *vocabularies*. A vocabulary provides the semantic information in the form of attribute-value pairs. The values may be single values, numeric ranges, or lists of acceptable values. A vocabulary doesn't describe anything; it is just a particular way of describing things. A *description* assigns values to the attributes of the vocabulary. The analogy is to objects. The class corresponds to the vocabulary; the instance, to the description.

A vocabulary consists of a set of keywords called *attributes*. Each attribute has a number of properties. More detail on each is given in the sub-sections that follow.

1. Name: A string different from that of any other attribute in this vocabulary.
2. Type: The value type of any description using this attribute, *e.g.* integer, string.
3. Multiplicity: One of `single`, `multiple`, `range`.
4. Required: Set to `true` if at least one value must be supplied for this attribute.
5. Must-match: Set to `true` if a look-up will succeed only if this attribute matches.
6. Matching rule: Defines what constitutes a match.
7. Designation: Ordered or Unordered.

The vocabulary developer decides on these properties. Anyone using the vocabulary, either to advertise a service or to discover one, must follow the rules of the vocabulary being used.[1]

2.1 Value Type

Although there is no theoretical reason for it, all values specified for a particular attribute must be of the same type.[2] All the basic types – string, int, float, *etc*., are supported. In addition, new value types can be introduced as long as their matching rules are expressed

[1] This decision is arbitrary, at least for some of the properties. For example, it might make sense for the advertiser to decide what constitutes a match. Such a change would have no impact on the work being reported here.

[2] Allowing mixed value types would not affect the negotiation strategy.

in terms of the base types.[3] For example, a value type of `Bigint` can be defined as consisting of two ints.

2.2 Multiplicity

Some attributes may only take on a single value. For example, when listing availability, it makes sense to advertise an item as in stock or not, but not both. Other times, it is reasonable to specify a set of values. A shoe seller may want to list all colors available for a particular style.

2.3 Required

Some attributes must be specified or there's no point in advertising or looking up the service. For example, the vocabulary developer may decide that there is no point in negotiating unless payment method is included in the advertisement.

2.4 Must-match

It is often the case that there is no point in reporting a match if a particular attribute was not included in the constraint expression, *e.g.*, payment method. After all, if we can't agree on means of payment, there's no point in trying to negotiate. Such an attribute can also be used as a simple form of password protection. If you don't know the password, you can't find the service.

2.5 Matching Rule

Many systems provide flexibility in the matching rules, but these rules are invariably based on the type of the attribute.[4] Often this approach is too limiting. For example, a shoe's width and size may both be represented by integers, as done in Europe, but the vocabulary developer may know that width requires an exact match while sizes that differ by one unit may still fit. Hence, attribute based matching rules allow more flexibility.

2.6 Designation

Every attribute is designated either *ordered* or *unordered*. If the type is ordered, we assume that the attribute values are specified in preference order, from most preferred to least preferred, but there is no indication of the strength of the preference. The order in which unordered attribute values appear does not indicate a preference.

The distinction between ordered and unordered attributes corresponds loosely to cooperative attributes and competitive ones. We say that some attributes are cooperative because the both parties may gain for certain values. For example, I may want to buy wingtips, but the shoe store benefits if I buy either wingtips or loafers. In this case, we both benefit by finding an acceptable value for the attribute. We assume that ordered

[3] This feature has not yet been implemented.

[4] LDAP does have per-attribute matching rules, but all attributes values are strings.

attributes are cooperative, in the sense that the preference order of the parties is not conflicting. Unordered attributes are those for which we assume there is conflict.

There are also cases that are not as clear-cut. For example, the store may not have the color shoes I want, yet they may provide them, say by buying them from a competitor. In this case, color becomes a competitive attribute because the store may lose money on this deal. Clearly, knowledge that such a case exists can affect the negotiation strategy. It might also be the case that the vocabulary designer doesn't know if a particular attribute is cooperative or competitive. Delivery time may be cooperative, if both parties benefit from a shorter delivery time, or competitive if one benefits and the other is penalized. I could, therefore, introduce an *unknown* designation, but there doesn't appear to be a practical way to make use of this idea in developing a negotiation strategy.

3 Contracts

Reaching a deal is the point of any negotiation; a *contract* is the specification of that deal. For this work, I'll assume that the parties are negotiating in some *marketplace*. A marketplace is defined by a set of rules, including such things as who can participate. One of the things I'll assume a marketplace provides is a set of *contract templates*. Each contract template is made up of one or more *sections*. Each section is expressed in terms of a specific vocabulary. The contract is a template because the values of the attributes in the vocabulary of each of the sections are not determined; that is the goal of the negotiation. This framework is not unique to this work. [9]

A typical marketplace will provide templates made up of different sections, each section representing an aspect of a deal. For example, there will almost certainly be a section on payment method and another on delivery terms, as well as one or more specific to the product. Thus, a shoe-buying marketplace would specify a section with a vocabulary including attributes for size, width, style, material, *etc*. There may be other sections, as well, including, perhaps, sections for return policy and warranty terms.

Clearly, in the most general case, we need a negotiation to determine which sections are included in the contract template. Since this negotiation is most likely identical to the negotiation for the product itself, we'll assume the marketplace specifies the sections. This assumption may not be warranted, making this topic a possible area for further work.

The terms specified by a section are represented by a vocabulary. Any of the attributes in the vocabulary may be included in the negotiation. Terms that are not included are considered irrelevant, although the marketplace may define default values. For example, shipping cost need not be included in the negotiation if I'm taking the shoes home with me. Also, delivery time may be set to a default of zero, indicating that the customer will take the merchandise at the time of purchase. Negotiation on this term is only needed if its value is to be changed.

4 Negotiation Protocol

The negotiation protocol used here consists of a number of steps. First, the product is advertised by specifying attributes in a vocabulary or set of vocabularies specified in a particular contract template. Next, a potential customer finds the advertisement by performing a look-up based on a constraint expression built up using attributes from the vocabulary or vocabularies. Finally, the parties negotiate the terms of a contract by agreeing to specific values of attributes in the contract template. The last step involves committing to these terms in a contract.

The protocol presented in Section 4.2 has a number of goals. First of all, it is designed to reduce the likelihood of a negotiation failing. Secondly, it is guaranteed to terminate in a finite number of steps with each party requiring only a modest amount of information. There is no need to use heuristics, such as the number of rounds or amount of time, and no need to worry about cycles, returning to an offer that was made before.

4.1 Finding Possible Deals

We assume that a particular item is advertised in a particular set of vocabularies, and that the item is found by specifying a constraint expression in those vocabularies. For example, a contract template for buying shoes may have a shoe vocabulary with attributes for style, color, and manufacturer; another vocabulary specifying payment method and price; and a third for delivery options. A search may ask for black or brown wingtips in size 10 to be paid for by Visa or MasterCard from a merchant who has the shoes in stock.

This simple example shows some of the key elements of the negotiation. First, there are a finite number of attributes in the contract template, only some of which are used for the advertisement or the look-up. Those terms not included in either may not be added to the negotiation, and the corresponding attributes are effectively removed from the contract template. If an attribute is not in either the advertisement or the lookup, we assume neither party cares about it. For example, a shoe store in a shopping mall expects customers to walk out with what they buy; customers of the store expect the same. Thus, even if the contract template specifies clauses for shipping, neither party cares to negotiate their values. The marketplace determines suitable default values.

4.2 Two-Party Protocol

The protocol deals with two parties that we'll call the *listener* and the *initiator*. Normally, the seller advertises wares and waits to be approached by a potential buyer. In this case, the seller is the listener and the buyer is the initiator. However, it may be that the buyer has advertised a need, say in a Request for Proposal. In this case, the roles are reversed.

The protocol involves the following steps.

1. The listener advertises in the vocabularies of the contract template.
2. The initiator does a lookup by specifying a constraint expression in these vocabularies.

3. The initiator send to the listener an offer consisting of values, often numeric ranges or sets of values, for a (proper) subset of the attributes used in the lookup.
4. The listener sends to the initiator a counteroffer consisting of values, often numeric ranges or sets of values, for a (proper) subset of the attributes used in the advertisement.
5. Either party may send a *negotiation failed* message at any time.

An attribute is *settled* if it appears with a single value in an offer and the same value in the counter to that offer. A settled attribute becomes a clause in the contract and is binding should all the terms in the contract template be settled.

A counteroffer is valid if it satisfies a few conditions.

1. Any attribute that was included in the advertisement (lookup) may be introduced by the listener (initiator) at any time subject to rule 5.
2. The numeric range or set of values of at least one attribute must be narrowed, or a new attribute introduced.
3. An attribute is removed from the negotiation once its value has been settled.
4. No attribute that has been settled may be reintroduced into the negotiation.
5. If all the attributes from a section that were introduced into the negotiation are settled, no more attributes from that section may be introduced.

A section is said to be *closed* once all its attributes that have been included in the negotiation are settled.[5] An invalid offer results in a failed negotiation.

4.3 Dealing with Attribute Types

Ordered and unordered attributes are handled somewhat differently. Revealing preferences of competitive attributes can put you in a bad bargaining position, but that is not the case for cooperative attributes.[6] We use this fact to accelerate the negotiation by assuming that cooperative attributes are rank ordered and competitive ones are not. For example, a specification "style = wingtips, loafers sandals" is assumed to be ordered while "quality = low, medium, high" is not. The strength of the preference is not indicated, and misrepresentations are difficult, but not impossible, to detect.

There are special rules for attributes with numeric types that depend on whether the attributes are ordered or unordered. Ordered numeric attributes are represented by a numeric range. For example, if I want to buy in bulk, I might want no more than 500 pairs of shoes, and I don't want to bother with buying unless I can get at least 100 pairs. In addition, I can list my preference since there's no reason the smaller value must come first. Similarly, the seller might have different terms and conditions for different size orders. For example, the minimum order size might be 250 pairs and a maximum size 1,000 pairs. On each round, either party may choose to narrow the range until agreement is reached on a specific value.

---

[5] This simplification is not strictly needed to meet the goals of the protocol. However, closing sections reduces the number of attributes that may be introduced into the negotiation, thereby simplifying the analysis needed by the strategy.

[6] A man walks into a shoe store and tells the clerk he'd like to buy a pair of shoes. "Loafers or wintips?", asks the clerk. "I'm not telling you.", says the customer. "It will put me at a competitive disadvantage".

Unordered attributes of numeric types are handled differently. There is no point is specifying a range on such an attribute.[7] For example, we know the buyer wants the lowest possible price, and the seller, the highest. In fact, giving a range would be revealing the least acceptable value, giving too much information to the other party. Hence, for unordered attributes of numeric type, one party introduces a single value, and the other does the same, either higher or lower. These first two values specify a range that must be narrowed on subsequent rounds. So, the buyer might offer $50 per pair, and the seller could counter with $100. On subsequent rounds, either party can specify a new value, as long as it narrows the existing range. For example, the initial offer might be

1. Style=wingtip,loafer; Color=black;brown; Quantity=100-500; Price=50

And the counter offer might be

2. Style=wingtip; Color=black; Quantity=500-250; Price=50:100

Note that the seller's range is 250-1,000, but the quantity in the counter offer can't widen the specified range. Subsequent rounds might be

3. Style=wingtip; Color=black; Quantity=500-250; Price=50:100
4. Style=wingtip; Color=black; Quantity=400-400; Price=70:100
5. Style=wingtip; Color=black; Quantity=400-400; Price=70:70

There is nothing in the rules that prevents one party from changing the other's offer. For example, if the price range is 50:100, the seller could offer 60:100. In effect, the seller is asking if the buyer will raise the bid. This strategy may not be a good one, since the buyer may declare a failed negotiation rather than let the seller control the offer in this way.

4.4 Disjunctions

While what has been described can be used, it is not expressive enough. Look at the style and color from the example. The seller might have only black wingtips and brown loafers, but the protocol does not provide a way to say that. We might go quite far in a negotiation before finding out that the buyer wants black loafers.

The solution is to allow *disjunctions*. In the above example, the first counteroffer could be

2. Style=wingtip; Color=black; Quantity=250-300; Price=50:90
   Style=loafer; Color=brown; Quantity=400-500; Price=50:70

Each disjunction must separately follow the rules spelled out in Section 4.2, including being declared failed. A deal is reached when all the terms in one disjunction have been agreed upon, and only one disjunction is left.

Why not just try black wingtips first and then brown loafers if the first one fails? That would be inefficient, particularly if there's a substantial cost for a negotiation.

[7] There is an implicit assumption here that the dependence is monotonic. While true for price, there is no reason this assumption must be valid. However, this assumption is not unreasonable, and making it simplifies picking a counteroffer.

Disjunctions allow multiple negotiations to be carried out at the same time, amortizing the costs.

4.5 Summary of Protocol

We can now see why the negotiation terminates in a finite number of steps. There are a finite number of sections in a contract template, and a finite number of attributes in the vocabulary for each section. Each step removes at least one attribute value or narrows a numeric range. Each party need only remember the previous offer to compare with the current counteroffer to determine if the rules are being followed.

Numeric ranges appear to be a problem. After all, I can increase the price I'm willing to pay by $0.01 on each round. While the negotiation will end in a finite number of steps, the number may be so large as to be effectively infinite as far as the parties are concerned. While we could impose heuristics on the amount by which numeric ranges must be narrowed, we rely instead on the behavior of the parties. We assume negotiations carry a cost that depends on the length of the negotiation. Should either party not negotiate in good faith, the other can always declare a failed negotiation.

This protocol has some shortcomings. For example, you can't change your mind. You may have agreed to buy a pair of Florscheim's, but the seller wants more money than you're willing to pay. With a more general protocol, you could change the manufacturer to a less expensive brand, such as Dexter. This protocol doesn't give you that option. You must declare a failed negotiation and start over.

This rule means that you must save more than just the previous offer and current counteroffer to avoid cycles; you must also remember what negotiations failed. I am supposing that all you need remember is the list of offers that led to the failure. This hypothesis needs to be verified. If you must remember the entire history of failed negotiations, the arguments about the amount of state needed are wrong.

One unfortunate effect of the strict rules of this protocol is that there may be deals that are possible that you don't find. We know that there are strategies almost certain to miss deals. For example, *take it or leave it*, in which the offer consists of a single value for all attributes, is likely to miss many possible deals. One motivation for the strategy described in Section 6 is to find a way to guarantee to find a deal should one exist.

The advertisement specifies a space of available products; the search, a space of desired products. Due to limitations in the expression of advertisement and look-up, however, we're not sure if a deal exists.[8] Thus, all we know is that there is the *possibility* of a deal. It's analogous to forming the convex hull of the points defining a non-convex polygon. Every point in the polygon is inside the convex hull, but all points inside the convex hull

[8] We could always list every possible deal, but the combinatoric explosion makes this strategy impractical except for examples of modest size.

are not inside the polygon. Because of this analogy, we'll use the term *convex hull offers* to denote offers with multiple values for attributes.[9]

5 Evaluating an Offer

Implicit in the concept of negotiation is an evaluation function that assigns values to different deals. For example, I am likely to value a pair of Florscheim shoes more highly than Dexter. The most common representation for this fact is a *utility function* [*6*]. Each of the parties to the negotiation prefers deals with larger values of its utility function. Thus, competitive attributes decrease the utility of one party and increase the utility of the other. Cooperative attributes affect the utilities of the participants in the same direction, either positive or negative. This fact doesn't mean that the parties will automatically agree to values for cooperative attributes; one's utility function may be maximized for one set of attribute values, and the other's maximized by a different set of values.

5.1 Conventional 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.[10] A deal consists of a single value for each of the *J* attributes. Each of the negotiating agents *i*, $i \in \{listener, initiator\}$, has a value $V_i(x_{1,k_1}, \ldots, x_{J,k_J})$ representing its utility. This general formulation is normally replaced with a linear combination of terms, namely

Equation 1

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

where $V^i(x_{jk})$ is the value negotiator *i* assigns to the *k*'th value of attribute *j*, and w_j^i is a weight function representing the relative importance of the *j*'th attribute to agent *i*.

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 zero otherwise. Only one of the elements of $\vec{b}_j$ will be 1 if we are dealing with a complete offer.[11] The k'th element of the vector $V^i(\vec{x}_j)$ has the value of the corresponding attribute. Hence, the summation over k is an inner product of

[9] The term *cross product offers* is another possibility, but it applies more to attributes with discrete values. Convex hull seems to be more general in that the handling of continuous variables is more natural.

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

[11] This factor is usually omitted in the definition of the utility function, because most people consider only a single value for each attribute.

these two vectors.[12] The result is the value assigned to the element of the *j*'th attribute that appears in the offer.

5.2 Evaluating Convex Hull Offers

If we are dealing with a convex hull offer, more than one value of each attribute may be included in the offer. In order to properly represent the utility, we modify the definition of the utility. 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 its utility. Constraints are easily represented in this form using a penalty function; the value $V^i(x_{jk})$ is set to $-\infty$ if the attribute value violates a constraint. We also change the two values that appear in the binary vector $\vec{b}_j$ 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 valuation, 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[13]

Equation 2

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

As we noted in Section 4.2, it is not necessary for an attribute to be introduced into the negotiation at all. However, unless the attribute is *required* as described in Section 2.3 we can't simply take $b_{jk} = -\infty$ for all values of this attribute, or the utility would be $-\infty$, indicating that no deal is possible. We deal with this fact by assuming there is a value for each attribute representing its absence in the offer. The value is always zero. 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 vocabulary designer, as described in Section 2.3. We only need to choose no deal over one with a zero utility to make this formulation work in all cases.[14]

---

[12] Although b_{jk} and $V^i(x_{jk})$ both have two indices, we don't treat them as matrices for two reasons. First of all, each attribute has a different number of values, which would result in a ragged matrix unless we padded it. More importantly, we are only interested in the diagonal of the product matrix.

[13] We can simplify the notation by using a generalized inner product, one that does a pairwise addition of the elements and uses max as the reduction operator. Doing so gives $V_i = \sum_{j=1}^{J} w_j^i \vec{b} \oplus V^i(\vec{x}_j)$.

[14] 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.

5.3 Composite Utility Functions

While more tractable mathematically than the more general form, this simplification 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.[15,16]

The solution adopted here is to use a utility function of the form of Equation 2 for each enumerated set of attribute values. In other words,

Equation 3 $$V_i = \sum_{j=1}^{J} w_j \max_{k \in C} \left[b_{jk} + V^i(x_{jk}) \right] + \sum_{j=1}^{J} w_j \max_{k=1}^{K_j} \left[b_{jk} + V^i(x_{jk}) \right],$$

where the first term is over the various combinations and the second term is over the remaining attribute values. The vector $\vec{x}_c$ represents the set of attribute values x_{jk} appearing in the combinations. For example, a component of this vector might stand for color=black and style=wingtips. Note that we must use combinations to represent something as simple as a different price for wingtips and loafers.

We need to be careful. If one of the relevant attribute values or combinations doesn't appear in the convex hull 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 second summation is red shoes, which I do not want, it will appear that no deal is possible, even though black and brown may be in the offer. The same applies for the combinations. I may be willing to buy red shoes if I can't get black wingtips or brown loafers. If I'm not careful, though, the first term will contribute $-\infty$. The solution is to make sure that each of the terms in Equation 3 includes an entry with zero value if none of the options is included in the offer as described in Section 5.2. 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 3 because we're enumerating the combinations. However, we need this factor to allow us to separate different combinations. For example, I prefer black wingtips to brown loafers, but I also prefer to pay cash for immediate delivery and by

---

[15] 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.

[16] 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

$w_1 x_1 + w_2 y_1 > w_1 x_2 + w_2 y_2$

$w_1 x_2 + w_2 y_2 > w_1 x_1 + w_2 y_2$.

$w_1 x_2 + w_2 y_2 > w_1 x_2 + w_2 y_1$

The second equation implies that $w_1(x_2 - x_1) > 0$, and the third, $w_2(y_2 - y_1) > 0$. Thus, $w_1(x_2 - x_1) + w_2(y_2 - y_1) > 0$, which contradicts the first equation.

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 combination. 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.

5.4 Price Equivalence

One problem with the linearized form of the utility function is that if price appears as an attribute, the utility function reduces every option to a monetary value. More generality is needed. Not every attribute value can be associated with a price[17], even if we allow negative prices. 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 a price at which I will pay cash for later delivery, just not a reasonable one.

This work is based on the presumption that no price offer will be unreasonable, even allowing for negative prices. The object 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 negative prices as long as it doesn't exceed the disposal costs. [18] 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 for my first choice is on the table that doesn't violate my constraints.

---

[17] "I wouldn't take that if they paid me."

[18] Here, we are making the reasonable assumption that a negative price will not exceed the disposal cost of the item. If I offer you $1,000,000 to take my left shoe, you'll certainly agree. If I offer you $1 to take 5,000 used left shoes, you'll certainly decline. The difference is that the disposal cost exceeds the amount you're being paid in the second case but not the first.

5.5 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. Hence, utility is more properly specified as a partial order of the possible contracts [8]. 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. Of course, such an approach precludes the use of the tools of mathematical optimization.

While Equation 3 avoids combinatoric explosion, it has a problem for convex hull deals. Namely, Equation 3 does not address the issue of rank ordering the deals. However, a simple modification allows us to solve this problem. We write

Equation 4 $$\vec{V}_i = \left[\vec{b}_r + \vec{V}^i(\vec{x}_r)\right] + \sum_{j=1}^{J} w_j \max_{k=1}^{K_j}\left[b_{jk} + V^i(x_{jk})\right] + \sum_{j=1}^{J} w_j \max_{k=1}^{K_j}\left[b_{jk} + V^i(x_{jk})\right],$$

where the set $r \in R$ 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, we can't use the form of Equation 4 in our strategy described in Section 6. As we'll see, that strategy assigns a probability of reaching any potential deal. In order to select a good counteroffer, I have to 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.

5.6 Finding the Worst Feasible Result[19]

There are times when it is convenient to bound the range of values of an offer. For example, if the best possible outcome from one disjunction is worse than the worst possible outcome of another, there is no need to consider the worse offer. This fact doesn't mean we should declare the worse disjunction failed. The other party may declare the better one failed at some time in the future. Instead, it means that we should focus our efforts on the better disjunction, keeping the worse one as a fallback position.

A simple change to the definitions used for Equation 2 allows a simple computation of the worst possible result. If we say that an attribute value that violates our constraints has a value of $+\infty$, and the value of the component of the vector $\vec{b}_j$ corresponding to an

[19] I haven't found a need for the result of this section yet.

attribute value that is not included in the offer also has a value of $+\infty$, then Equation 2 and its generalizations provide a convenient way to compute the minimum possible value if we take the minimum instead of the maximum of the various terms.[20]

5.7 Attributes with Continuous Values

We said in Section 5.1 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, ordered 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.[21] 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 $H(0, 200 - P, \infty) = H(-200, -P, \infty)$ to represent the contribution of price to utility.[22]

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.

5.8 Why Separate Deals

It's easy to see how we could be misled by defining a utility function that combines the preferred deals, *e.g.*, $V_{buyer} = V^1_{buyer} + V^2_{buyer}$ or any other simple function. 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[23], but I do care for brown loafers. Any 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 change should have no effect on my utility for black wingtips, but it does if we combine deals in a simple formula. This affect is handled by combining delivery with style and color into a composite term.

---

[20] Of course, operationally, all we need do is turn every $-\infty$ into a $+\infty$, but having a formal expression guides an understanding of the behavior.

[21] There is no theoretical reason that requires the limits to be constants. They can be arbitrary functions of time. We do not allow them to be functions of the values of other attributes, though. That effect is handled using combinations.

[22] The sign of *P* must be negative to reflect greater utility from lower price. A cut-off of ∞ implies that the buyer is willing to be paid to take the merchandise, *i.e.,* accept a negative price.

[23] My black suit is at the cleaners.

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 in the list of combinations, or even as terms in the sum over individual attribute values. In this case, they contribute, say

$$V_{buyer}(wingtips, brown) = [2w_1 + w_2]$$
$$V_{buyer}(loafers, black) = [3w_1 + 2w_2] \quad ,$$

where w_1 and w_2 are the relative weights for style and color, and their coefficients represent the value assigned to each value. Providing such terms in the utility function is equivalent to stating that we would purchase one of these configurations given the right circumstances.

5.9 Accounting for Time

We also need to include the "monetary value of time". Actually, 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.[24] In this case, we subtract a nonlinear term, which may be different for each deal. We may also change the relative values of various attribute values or their combinations, so we make $V_i(x_{sk})$ 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 $V_i(x_{rk})$, 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.

[24] The silver pumps are worth very little to the coed the day after the prom.

5.10 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 represented as arbitrary functions. Fortunately, the prototype is written in Python, which supports the ability to execute a string representation of a function.

The representation I settled on is a 2-element tuple. The first element describes the time dependence of the cost of the negotiation. The second is a tuple of valuations of combinations in which individual attributes are considered to be combinations with one term. Each entry in the second term is a 3-element tuple. The first component of the combination tuple is the weight given the combination; the second, a tuple consisting of the specific contributions to the combination; and the third, a function representing the value. In more formal terms we have

Valuation:: (timedep, combinations)
Combinations:: combination | (combinations)
Combination:: (weight, attributes, values)
Weight:: Integer
Attributes:: attribute | (attributes)
Attribute:: (vocabName, attName, values)
VocabName:: string
AttName:: string
Values:: [value] | [values]
Value:: string interpreted as an executable expression | constant
TimeDep:: value
Constant:: string | integer | float

An example will illustrate the features of this representation.

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

```
    ((((shoe','color', ['brown']),),(1,0)))))
  (1,-INFINITY,(
    ((('delivery','delay',[0,0]),),(5,0)),
    ((('delivery','delay',[1,1]),),(2,0)),
    ((('delivery','delay',[2,2]),),(1,0))))
  (4,0,(
    ((('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 that the extra valuation , the number immediately following the weight, is zero, which means that other combinations will be considered. The value of $-\infty$ in the delay/method combination ensures that no other combination will be accepted.

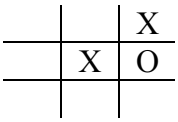
6 Constructing a Counteroffer

Your agent has just presented my agent with a convex hull offer. My agent must construct a counteroffer, preferably one that leads to a good end result for me. This section describes a procedure for selecting a counteroffer similar to that used by computers when playing games, such as chess. The problem here is considerably more complex in some ways, as we'll see.

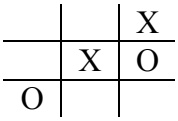
6.1 Playing a Game

Computer programs that play games don't attempt to emulate the way a person thinks. Instead, they use their raw computational power to try every possible move, every countermove, every response to that, and so on, until they have followed every possible path or run out of time. Full exploration of the game tree is often made unnecessary by applying symmetries and pruning the tree. One pruning strategy says that I don't need to explore a subtree if the best possible result on that subtree is no better than the worst possible result on some other subtree. Assigning values to the best and worst possible results requires knowledge of the game, and the evaluation function is a key factor in what makes one program better than another.

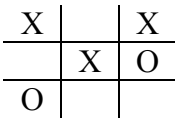
Consider a simple example, tic-tac-toe. Because of symmetry, there are only 3 starting moves, center, corner, or edge. If X is placed in the center, there are only 2 responses, corner or edge, again because of symmetry. If O is placed on an edge, there are 4 responses, one of which is a corner adjacent to the O. Our exploration of the game tree has brought us to the position



There are six possible responses for O, five of which lead to an automatic loss for O. Hence, O explores moving to the lower left corner.



Of X's five possible responses, one of them is the upper left corner.



No response by O can prevent X from winning on the next move. A reasonable heuristic is that any time the opponent has two winning moves the game is lost, so O need not explore any further. In other words, the best possible result from this position is a loss, while the worst possible result from any other position is a loss, but may be a draw or a win. Since every possible response to X's second move has led to a loss, O's first move is to a corner.

Tic-tac-toe is a trivial game; you can enumerate the entire game tree on paper in a few minutes. Chess is more complex, but the basic idea is the same. At some point in exploring the game tree, you find a position that you can't win, your King *versus* King and Queen. There's no point in exploring that part of the game tree, since you'll make a move to avoid that position if you can. Less extreme cases require more sophisticated heuristics. Is a Queen worth two Rooks? Are you better off trading a Bishop for a Knight if the exchange damages your opponent's Pawn formation? If you could continue to explore the game tree, you could answer these questions, but there are far too many possibilities. Instead, you apply heuristics to decide which lines of play to examine, and do the best job you can to evaluate positions reached at the end of your search.

6.2 Negotiation as a Game

The negotiation protocol described in Section 4 is guaranteed to terminate in a finite number of steps. Hence, we can view a negotiation as a finite game and expand the game tree as described in Section 6.1. However, there are some complications that don't appear in conventional games.

The first complication is the fact that either party may introduce a new attribute into the negotiation at any time, and there are no restrictions, other than those imposed by the marketplace, on the values of that attribute. For example, I may have been negotiating

the price of black wingtips when the seller introduces a shipping time attribute. The seller may be able to offer a lower price if I'm willing to wait a week for a shipment directly from the factory. I may have a problem if I've been traversing the game tree along paths that include only price, a color attribute with a value black, and a style attribute with a value wingtip. In trying to find a path that leads to a good result for me, I may have led the negotiation to a place that excludes the best possible deal had I considered waiting for delivery.

Another complication is that I don't know what the other party wants. In chess, there is one utility function; each side wants to win but will settle for a draw. When examining nodes in the game tree, I know what you are trying to do and can use that information in predicting how you will respond to my moves. In a multi-attribute negotiation, I don't know your utility function, so I can't be sure what direction you'd like the negotiation to take. I do know that the sign of the coefficients of unordered attributes in your utility function is the opposite of mine. I also know your preferences for the ordererd attributes. I don't know the relative weights of the terms in your utility function.

These complicating factors make the game one of uncertain information. In such situations, all I can do is estimate the missing information based on the available data. The better my data, the better I will do in predicting your counteroffers.[25] I'll model these uncertainties as probability distributions. In essence, I'll adjust my evaluation of a leaf of the tree by considering a lottery over your moves at each level of the tree. The result will be a probability that the value of the leaf will be achieved.

In game theory, each player is assigned a *type* that denotes the choice the player will make on each turn. Hence, each type is associated with a complete path through the game tree from root to leaf. Even if two paths reach the same position, a *transposition*, the types of the two players are different because they made different choices to reach the common position. The assignment of the probability of the opponent making a particular move is equivalent to the probability that the opponent is of a particular type.

We will assume that we know the distribution of types, but we don't know the specific type of the player for a given negotiation. This assumption can be relaxed if there is a history of negotiation between two parties. We will update our estimate of the player's type as the negotiation proceeds using information gleaned from the counteroffers. For example, a risk averse opponent will counter price offers differently than will one who is risk neutral. Developing algorithms for refining the update procedure is expected to be an ongoing activity.

6.3 Making Use of Disjunctions

There are some very simple situations that can lead to conundrums that the agent may not be able to resolve. For example, consider the shoe example with the partial order

[25] Bayesian updates may enable me to make more accurate predictions of your behavior if there is a history of negotiations.

Black wingtips > brown loafers
Brown loafers > black loafers
Brown loafers > brown wingtips

Say that I've just received the convex hull offer

Style = wingtips, loafer; Color = black, brown

My counteroffer must narrow the space. I really want black wingtips, but if I eliminate either loafers or brown, I'll rule out my second choice. The solution is to carry on two, separate negotiations in parallel, one for black wingtips and one for brown loafers. If the first negotiation fails, I may still get my second choice.

The solution is to use disjunctions, but which ones? My counteroffer can be the disjunction

Style = wingtips; Color = black
Style = loafers; Color = brown

but this choice doesn't give me the option of settling for black loafers. On the other hand, either

Style = wingtips, loafer; Color = black
Style = wingtips, loafer; Color = brown

or

Style = wingtips, loafer; Color = black
Style = loafer; Color = black, brown

does keep all my options open.[26] We see that either of these choices leaves all acceptable deals on the table. If black wingtips are not available in the first pair, the first disjunction includes black loafers, and the second includes brown loafers and brown wingtips. In the second pair, the first disjunction leaves black loafers in the first term if black wingtips are not available. Black or brown loafers are covered by the second term. Clearly, any set of disjunctions must be selected to cover all possible deals that meet the constraints.

7 Sample Negotiation

This section steps through a simple negotiation to illustrate how the procedure works. The intent is to make the example small enough that the game tree can be enumerated on paper, not to produce a realistic case. Even so, the level of detail is well beyond the tolerance of most readers. Feel free to skim this section, particularly Section 7.7. Don't skip it entirely, though, as it shows how several important issues get resolved.

[26] Note that the four disjunctions is a full enumeration of every possible deal, something that is possible with this simple example but not in general. If I could enumerate all possible deals, then I'd be submitting a list of ultimatums, and the negotiation would reduce to each party eliminating one or more disjunctions on each round. This approach is more like strategies found in the literature.

7.1 Contract Template

We will assume a very simple contract template having only three sections. The first section will describe the product; the second, the payment method; and the third, delivery options. The goal is to show the linkage between attributes in different sections of the contract.

7.2 Vocabularies

Each section of the contract template requires a specific vocabulary. For this example, we'll use very simple vocabularies.

The Shoe vocabulary is shown in Table 1. It includes two of the three types of attributes. The manufacturer attribute was included as an attribute that doesn't appear in the negotiation.

Table 1. Shoe Vocabulary

Attribute	Value Type	Multiple	Required	Must Match	Matching Rule	Type
Style	String	Yes	Yes	Yes	Any = =	Ordered
Color	String	Yes	Yes	Yes	Any[27] = =	Ordered
Manufacturer	String or *[28]	Yes	No	No	Soundex[7][29]	Ordered

In Table 2 we show almost the simplest payment vocabulary possible. The type of the payment method is ordered because the seller has costs for each type of payment and would like to provide preferences. We're familiar with the fees charged by credit card companies, but check validation services also charge the merchant. What's less well known is that taking cash also has a cost. The obvious problem is theft, but another significant cost is the additional insurance premium charged for businesses that deal with large amounts of cash.

Similarly, we don't know the sign of the payment term in the buyer's utility function. Some buyers prefer to use a credit card because of the float; others want to avoid running up a large bill. Although it violates the agreement between the merchant and the credit card issuer, it is not unheard of for buyers to negotiate a reduced price in exchange for using cash because of the fee the merchant pays on each transaction. Again, the buyer would like to express preferences.

[27] We might also require that any number of values match, as when looking for two tone shoes.
[28] A value of * indicates that any value will match. In our example, it may be that the seller can order shoes from any manufacturer.
[29] This matching rule allows matches with misspellings using the Soundex algorithm.

Table 2. Payment Vocabulary

Attribute	Value Type	Multiple	Required	Must Match	Matching Rule	Type
Method	Set: Cash, Check, Visa	Yes	Yes	Yes	Any = =	Ordered
Price	Integer	No	Yes	Yes	One of: = =, <, >	Unordered

Table 3 is another very simple vocabulary. In declaring its one attribute ordered, we are assuming that both parties benefit with a shorter delivery time. Perhaps the buyer wants the merchandise as soon as possible, and the seller only gets paid once the goods are delivered. This attribute could also be unordered if, for example, the delivery charge is included in the price, and faster delivery costs more.

Table 3. Shipping Vocabulary

Attribute	Value Type	Multiple	Required	Must Match	Matching Rule	Type
Delay	Integer in 0-7	No	No	No	In range[30]	Ordered

7.3 Finding a Vendor

In Section 4.1 we saw that the negotiation starts with an advertisement and a lookup request. In our case, we'll assume that the seller advertises his wares and the buyer searches for them. Thus, the seller has the role of listener; the buyer, that of initiator.

Let's assume that the seller posts the following advertisement.

Shoe Vocabulary
    Style = sandal, loafer, wingtip
    Color = black, brown, tan
    Manufacturer = Florscheim, Dexter
    Price[31] = 50
Payment Vocabulary
    Payment = cash, Visa
Delivery Vocabulary
    Delay = 0-5

The buyer might specify a constraint expression

[30] The range specified in the advertisement must overlap the range specified in the search request.
[31] The interpretation of the single value specified for a numeric, unordered attribute depends on the context. If the value represents a price, it is the minimum (maximum) price if the advertiser is a seller (buyer).

Shoe Vocabulary
    Style = wingtip OR loafer
    Color = black OR brown
    Price < 300
Payment Vocabulary
    Payment = cash OR check OR Visa
Delivery Vocabulary
    Delay < 3

These two are compatible, so the buyer will find this seller and can enter into negotiation.

Haven't the buyer and seller revealed too much information? After all, the seller has specified a minimum acceptable price, and the seller has given a maximum. There are two reasons why this information doesn't present a problem. Firstly, in many middleware systems, the matching is done by a disinterested third party. In such an environment, the buyer doesn't see the advertisement, and the seller doesn't see the search expression. Secondly, these numbers need not be the actual bounds. By expanding the available range, both buyer and seller leave some additional room for negotiation since either or both may be willing to relax their constraints. Mutually anonymous lookups are also possible.[32]

Note that the advertisement gives us enough information to expand the game tree. However, the buyer's initial offer will certainly reduce the size of this tree significantly, so we'll delay this step until Section 7.6.

7.4 Utility Functions

Let's look at the utility functions for this example. The buyer's utility for black wingtips is 3, and 2 for brown loafers. The buyer doesn't have a preference between the other two combinations, so we can assign each one a value of 1. The buyer wants the lowest price, but won't pay more than $200; a higher price violates a constraint. Hence, the utility assigned to price is $200 - P$. The buyer also insists on paying cash if carrying the shoes out of the store and by Visa if they will be delivered. In this simple case, the buyer's utility is

$$\vec{V}_{buyer} = \max\left\{\begin{bmatrix} b_{black,wingtips} \\ b_{brown,loafers} \\ b_{absent} \end{bmatrix} + \begin{bmatrix} 3 \\ 2 \\ 0 \end{bmatrix}\right\} + \max\left\{\begin{bmatrix} b_{visa,delivery} \\ b_{cash,carry} \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix}\right\} +$$

$$\max\left\{\begin{bmatrix} b_{wingtips} \\ b_{loafers} \\ b_{absent} \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}\right\} + \max\left\{\begin{bmatrix} b_{black} \\ b_{brown} \\ b_{absent} \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}\right\} + H(-200,-P,\infty) - t\,.$$

[32] One example is the millionaire's algorithm that allows two wealthy people to decide which is worth more without revealing either's net worth.

We take H to be 0 if price is not included in the offer, in analogy to the way we handle the other attributes. Note that only the listed combinations of payment method and delivery are possible; all others lead to a failed negotiation.

If we assume the seller doesn't care what shoes he sells, prefers cash to credit regardless of delivery method, and wants the highest price that exceeds $100, then this utility is

$$V_{seller} = \max\left\{\begin{bmatrix} b_{wingtips} \\ b_{loafers} \\ b_{absent} \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}\right\} + \max\left\{\begin{bmatrix} b_{black} \\ b_{brown} \\ b_{absent} \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}\right\} +$$

$$3\max\left\{\begin{bmatrix} b_{cash} \\ b_{credit} \\ b_{absent} \end{bmatrix} + \begin{bmatrix} 2 \\ 1 \\ 0 \end{bmatrix}\right\} + 2\max\left\{\begin{bmatrix} b_{carry} \\ b_{delivery} \\ b_{absent} \end{bmatrix} + \begin{bmatrix} 2 \\ 1 \\ 0 \end{bmatrix}\right\} + H(100, P, \infty) - t\,.$$

In this example, price is unordered because its sign is different in the two utility functions, but the other attributes are ordered, even though there is a disagreement on the value of the payment conditions. You'll notice that the seller does not specify any combinations. Time has been included as a linear penalty term having a value of $1 per round.

7.5 First Offer

We can't determine these valuations without knowing the current offer. If the buyer offers

style=wingtips, loafers; color=brown,black; P = $125,

we get

$$V_{buyer} = \max\begin{bmatrix}3 & 2 & -\infty\end{bmatrix}^T + \max\begin{bmatrix}1 & 1 & -\infty\end{bmatrix}^T + \max\begin{bmatrix}1 & 1 & -\infty\end{bmatrix}^T + 75 = 80,$$

and

$$V_{seller} = \max\begin{bmatrix}1 & 1 & -\infty\end{bmatrix}^T + \max\begin{bmatrix}1 & 1 & -\infty\end{bmatrix}^T + 25 = 27\,.$$

These two valuations should not be compared to each other, because their relative scaling is arbitrary. Notice, too, that the attributes of payment and delivery are not included, so they contribute nothing to the utility.

7.6 Encoding Offers

The agents negotiate over the contract by sending changes to each other. These changes are in the form of a list of 3-element tuples. Each tuple consists of a vocabulary name, an attribute name, and a list of values. If the values do not obey the protocol, the negotiation is declared to have failed.

Thus far, we've been using a human readable representation. However, both this notation and the tuple form used in the software are too cumbersome to represent the expansion of the game tree. Instead, we'll use a compact notation and explain the essential points in text. The form chosen is an outline. At the highest level is the vocabulary. The next level will be attributes within each vocabulary. The third level will be for attribute values. Attributes with a numeric value type are represented by a range, which may be of zero size.

I Shoe
- A Style
  - 0 Absent
  - 1 Wingtips
  - 2 Loafers
- B Color
  - 0 Absent
  - 1 Black
  - 2 Brown
- C Manufacturer
  - 0 Absent
  - 1 Dexter
  - 2 Florsheim

II Payment
- A Method
  - 0 Absent
  - 1 Cash
  - 2 Visa
- B Price
  - 0 Absent
  - 1 Value range

III Shipping
- A Delay
  - 0 Absent
  - 1 Value range

7.7 Expanding the Game Tree

There is an offer on the table that the seller must respond to. As stated in Section 6.2, the next step is to expand the game tree by considering every possible counteroffer and all responses to each of them, *etc*. First, we'll look at all possible counteroffers to the buyer's initial offer. Completely specified deals will be shown in **bold**. We won't bother showing rejected offers. We'll start with an offer that includes a value for each attribute, and in Section 7.9 we'll describe how missing attributes are handled.

Normally, the buyer would not offer a price this early in the negotiation. However, delaying makes the hand expansion of the game tree more tedious. For the same reason, the seller's first counteroffer includes a price.

Why would the buyer offer so much? Perhaps to avoid violating the seller's price constraint. We can see from the seller's advertisement, that the minimum acceptable offer is $50, but the buyer doesn't know this value. The buyer must assign a probability that a given price offer will result in a failed negotiation. In this example, the buyer has judged that an offer of $100 is the lowest one with a sufficiently large probability of being acceptable to the seller.

	I			II		III
Offer	A	B	C	A	B	A
1	1,2	1,2	0	1	100	0:0

1.A[33]	**1**	**1**	**0**	**1**	**100:100**	**0:0**
1.B	**1**	**2**	**0**	**1**	**100:100**	**0:0**
1.C	**2**	**1**	**0**	**1**	**100:100**	**0:0**
1.D	**2**	**2**	**0**	**1**	**100:100**	**0:0**
1.E	1,2	1,2	0	1	100:200	0:0
1.F	1	1,2	0	1	100:200	0:0
1.G	2	1,2	0	1	100:200	0:0
1.H	1,2	1	0	1	100:200	0:0
1.I	1,2	2	0	1	100:200	0:0
1.J	1	1	0	1	100:200	0:0
1.K	1	2	0	1	100:200	0:0
1.L	2	1	0	1	100:200	0:0
1.M	2	2	0	1	100:200	0:0

Next, we'll look at the buyer's first round of counteroffers.

Offer	I			II		III
	A	B	C	A	B	A
1.E.1	1	1,2	0	1	100:200	0:0
1.E.2	2	1,2	0	1	100:200	0:0
1.E.3	1,2	1	0	1	100:200	0:0
1.E.4	1,2	2	0	1	100:200	0:0
1.E.5[34]	1	1	0	1	100:200	0:0

[33] This offer constitutes an ultimatum, since the buyer can only accept the offer or declare a failed negotiation. We won't show counteroffers to ultimata.

[34] This offer is not the same as 1.J because the seller must respond to this offer, and the buyer must respond to 1.J. Being the first to make a price concession is often disadvantageous.

1.E.6	1	2	0	1	100:200	0:0
1.E.7	2	1	0	1	100:200	0:0
1.E.8	2	2	0	1	100:200	0:0
1.E.9[35]	1,2	1,2	0	1	(100+x):200	0:0
1.E.10	1	1,2	0	1	(100+x):200	0:0
1.E.11	2	1,2	0	1	(100+x):200	0:0
1.E.12	1,2	1	0	1	(100+x):200	0:0
1.E.13	1,2	2	0	1	(100+x):200	0:0
1.E.14	1	1	0	1	(100+x):200	0:0
1.E.15	1	2	0	1	(100+x):200	0:0
1.E.16	2	1	0	1	(100+x):200	0:0
1.E.17	2	2	0	1	(100+x):200	0:0

1.F.1[36] 1.E.5

Nothing in the rules prevents the buyer from changing the upper bound to a smaller value, in effect changing the seller's offer. For the purpose of this hand expansion, we'll assume the probability of such an offer being accepted is zero. The code will have to include this possibility as an option.

The transpositions don't need to be expanded separately although they may affect the estimates of the opponent's responses to specific counteroffers. In game theory, we say that the *type* of the player is assigned at the start of the game, and the player's type determines the specific response. So, in this example, a player who responded to the initial offer with 1.E is a different type than one who responded with 1.F. Even though the set of legal moves from 1.F.1 is the same as from 1.E.5, the probabilities will change. Hence, the software that follows the tree must include the ability to update the estimations assigned to the probabilities of a response to include the path by which that node was reached. In our software, we'll use the labeling shown in the table to indicate

[35] This offer denotes that the buyer is willing to spend more than $100. In subsequent rounds, the value of *x* can only be increased.

[36] Selecting 1 for attribute B of vocabulary I gives an offer seen elsewhere in the tree. In games, we call such a situation a *transposition*. Since a substantial proportion of the counteroffers result in transpositions, they will be collected into a separate table and presented in a more compact form.

the player's type. Thus, one of our players will be of type 1.E.5 and the other of type 1.F.1.

Transpositions for Buyer's Response

1.F.1 = 1.E.5	1.F.2 = 1.E.6	1.F.3 = 1.E.10	1.F.4 = 1.E.14	1.F.5 = 1.E.15
1.G.1 = 1.E.7	1.G.2 = 1.E.8	1.G.3 = 1.E.11	1.G.4 = 1.E.16	1.G.5 = 1.E.17
1.H.1 = 1.E.5	1.H.2 = 1.E.7	1.H.3 = 1.E.12	1.H.4 = 1.E.14	1.H.5 = 1.E.16
1.I.1 = 1.E.6	1.I.2 = 1.E.8	1.I.3 = 1.E.13	1.I.4 = 1.E.15	1.I.5 = 1.E.17
1.J.1 = 1.E.14	1.K.1 = 1.E.15	1.L.1 = 1.E.16	1.M.1 = 1.E.17	

Now it's the seller's turn again.

Offer	I			II		III
	A	B	C	A	B	A
1.E.1.C	1	1,2	0	1	100:(200-y)	0:0
1.E.1.D	1	1	0	1	100:(200-y)	0:0
1.E.1.E	1	2	0	1	100:(200-y)	0:0
1.E.2.C	2	1,2	0	1	100:(200-y)	0:0
1.E.2.D	2	1	0	1	100:(200-y)	0:0
1.E.2.E	2	2	0	1	100:(200-y)	0:0
1.E.3.C	1,2	1	0	1	100:(200-y)	0:0
1.E.4.C	1,2	2	0	1	100:(200-y)	0:0
1.E.9.A	1	1,2	0	1	(100+x):200	0:0
1.E.9.B	2	1,2	0	1	(100+x):200	0:0
1.E.9.C	1,2	1	0	1	(100+x):200	0:0
1.E.9.D	1,2	2	0	1	(100+x):200	0:0
1.E.9.E	1	1	0	1	(100+x):200	0:0
1.E.9.F	2	1	0	1	(100+x):200	0:0
1.E.9.G	1	2	0	1	(100+x):200	0:0
1.E.9.H	2	2	0	1	(100+x):200	0:0
1.E.9.I	1,2	1,2	0	1	(100+x): (200-y)	0:0
1.E.9.J	1	1,2	0	1	(100+x): (200-y)	0:0
1.E.9.K	2	1,2	0	1	(100+x): (200-y)	0:0

1.E.9.L	1,2	1	0	1	(100+x): (200-y)	0:0
1.E.9.M	1,2	2	0	1	(100+x): (200-y)	0:0
1.E.9.N[37]	**1**	**1**	**0**	**1**	**(100+x): (200-y)**	**0:0**
1.E.9.O	**2**	**1**	**0**	**1**	**(100+x): (200-y)**	**0:0**
1.E.9.P	**1**	**2**	**0**	**1**	**(100+x): (200-y)**	**0:0**
1.E.9.Q	**2**	**2**	**0**	**1**	**(100+x): (200-y)**	**0:0**

Transpositions for Seller's Response

1.E.1.A = 1.J	1.E.1.B = 1.K	1.E.2.A = 1.L	1.E.2.B = 1.M	
1.E.3.A = 1.J	1.E.3.B = 1.L	1.E.3.D = 1.E.1.D	1.E.3.E = 1.E.2.D	
1.E.4.A = 1.K	1.E.4.B = 1.M	1.E.4.D = 1.E.1.E	1.E.4.E = 1.E.2.E	
1.E.5.A = 1.E.1.D	1.E.6.A = 1.E.1.E	1.E.7.A = 1.E.2.D	1.E.8.A = 1.E.2.D	
1.E.10.A = 1.E.9.E	1.E.10.B = 1.E.9.G	1.E.10.C = 1.E.9.J	1.E.10.D = 1.E.9.N	1.E.10.E=1.E.9.P
1.E.11.A = 1.E.9.F	1.E.11.B = 1.E.9.H	1.E.11.C = 1.E.9.K	1.E.11.D = 1.E.9.O	1.E.11.E=1.E.9.Q
1.E.12.A = 1.E.9.E	1.E.12.A = 1.E.9.F	1.E.12.A = 1.E.9.C	1.E.12.A = 1.E.9.N	1.E.12.A=1.E.9.P
1.E.13.A = 1.E.9.G	1.E.13.A = 1.E.9.H	1.E.13.A = 1.E.9.D	1.E.13.A = 1.E.9.P	1.E.13.A=1.E.9.Q
1.E.14.A = 1.E.9.N	1.E.15.A = 1.E.9.P	1.E.16.A = 1.E.9.O	1.E.17.A = 1.E.9.Q	

Here we see another simplification in the equivalence of offers from different rounds. If time appears in one or both of the utility functions, then the state of the world is not truly the same for offers made at different times. We can calculate the effect, though when the time enters the utility function linearly, as it does in this case.

The next round leaves only a few possible counteroffers.

Offer	I			II		III
	A	B	C	A	B	A
1.E.1.C.1[38]	1	1	0	1	100: (200-y)	0:0
1.E.1.C.2	1	2	0	1	100: (200-y)	0:0
1.E.1.C.3	1	1,2	0	1	(100+x): (200-y)	0:0
1.E.1.C.4	**1**	**1**	**0**	**1**	**(100+x): (200-y)**	**0:0**
1.E.1.C.5	**1**	**2**	**0**	**1**	**(100+x): (200-y)**	**0:0**

[37] This offer and the next three will be accepted deals when 100+x=200-y.

[38] As before, this offer is not the same as 1.E.1.D because the seller must respond to this one and the buyer to the other.

1.E.2.C.1	2	1	0	1	100: (200-y)	0:0
1.E.2.C.2	2	2	0	1	100: (200-y)	0:0
1.E.2.C.3	2	1,2	0	1	(100+x): (200-y)	0:0
1.E.2.C.4	**2**	**1**	**0**	**1**	**(100+x): (200-y)**	**0:0**
1.E.2.C.5	**2**	**2**	**0**	**1**	**(100+x): (200-y)**	**0:0**
1.E.3.C.3	1,2	1	0	1	(100+x): (200-y)	0:0
1.E.4.C.3	1,2	2	0	1	(100+x): (200-y)	0:0
1.E.9.I.5	1,2	1,2	0	1	(100+x): (200-y')	0:0
1.E.9.J.3	1	1,2	0	1	(100+x): (200-y')	0:0
1.E.9.K.3	2	1,2	0	1	(100+x): (200-y')	0:0
1.E.9.L.3	1,2	1	0	1	(100+x): (200-y')	0:0
1.E.9.M.3	1,2	2	0	1	(100+x): (200-y')	0:0
1.E.9.N.1	1	1	0	1	(100+x): (200-y')	0:0
1.E.9.O.1	2	1	0	1	(100+x): (200-y')	0:0
1.E.9.P.1	1	2	0	1	(100+x): (200-y')	0:0
1.E.9.Q.1	2	2	0	1	(100+x): (200-y')	0:0

Transpositions for Buyer's Response

1.E.1.D.1 = 1.E.1.C.4	1.E.1.E.1 = 1.E.1.C.5	1.E.2.D.1 = 1.E.2.C.4	1.E.2.E.1 = 1.E.2.C.5
1.E.3.C.1 = 1.E.1.C.1	1.E.3.C.2 = 1.E.2.C.1	1.E.3.C.4 = 1.E.1.C.4	1.E.3.C.5 = 1.E.2.C.4
1.E.4.C.1 = 1.E.1.C.1	1.E.4.C.2 = 1.E.2.C.1	1.E.4.C.4 = 1.E.1.C.4	1.E.4.C.5 = 1.E.2.C.4
1.E.9.A.1 = 1.E.14	1.E.9.A.2 = 1.E.15	1.E.9.A.3 = 1.E.1.C.3	
1.E.9.B.1 = 1.E.16	1.E.9.B.2 = 1.E.17	1.E.9.B.3 = 1.E.2.C.5	
1.E.9.C.1 = 1.E.14	1.E.9.C.2 = 1.E.15	1.E.9.C.3 = 1.E.3.C.3	
1.E.9.D.1 = 1.E.15	1.E.9.D.2 = 1.E.17	1.E.9.D.3 = 1.E.4.C.3	
1.E.9.E.1 = 1.E.1.C.4	1.E.9.F.1 = 1.E.2.C.4	1.E.9.G.1 = 1.E.1.C.5	1.E.9.H.1 = 1.E.2.C.5
1.E.9.I.1 = 1.E.1.C.3	1.E.9.I.2 = 1.E.2.C.3	1.E.9.I.3 = 1.E.3.C.3	1.E.9.I.4 = 1.E.4.C.3
1.E.9.J.1 = 1.E.1.C.4	1.E.9.J.2 = 1.E.1.C.5	1.E.9.K.1 = 1.E.2.C.4	1.E.9.K.2 = 1.E.2.C.5
1.E.9.L.1 = 1.E.1.C.4	1.E.9.L.2 = 1.E.2.C.5	1.E.9.M.1 = 1.E.1.C.5	1.E.9.M.2 = 1.E.2.C.5

All the deals in the final round of negotiation result in transpositions.

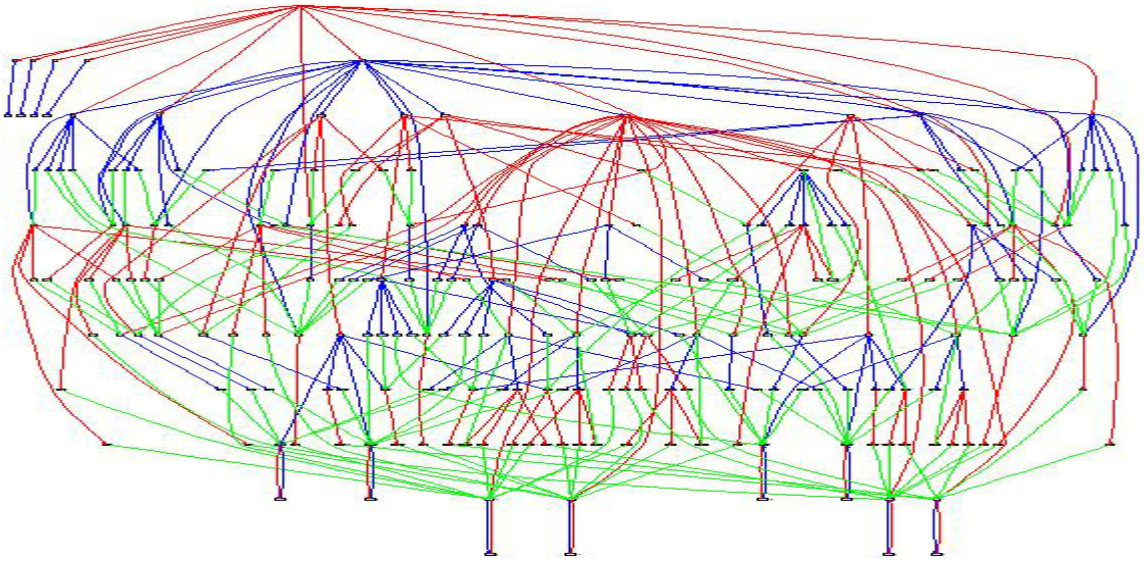
Transpositions for Seller's Response

1.E.1.C.1.A = 1.E.9.N	1.E.1.C.2.A = 1.E.9.O	1.E.1.C.3.A = 1.E.9.N	1.E.1.C.3.B = 1.E.9.P
1.E.2.C.1.A = 1.E.9.O	1.E.2.C.2.A = 1.E.9.Q	1.E.2.C.3.A = 1.E.9.O	1.E.2.C.3.B = 1.E.9.Q
1.E.3.C.3.A = 1.E.9.N	1.E.3.C.3.B = 1.E.9.O	1.E.4.C.3.A = 1.E.9.P	1.E.4.C.3.B = 1.E.9.Q
1.E.9.I.5.A = 1.E.9.J	1.E.9.I.5.B = 1.E.9.K	1.E.9.I.5.C = 1.E.9.L	1.E.9.5.D = 1.9.E.M
1.E.9.I.5.E = 1.E.9.I			
1.E.9.J.3.A = 1.E.9.N	1.E.9.J.3.B = 1.E.9.P	1.E.9.J.3.C = 1.E.9.J	
1.E.9.K.3.A = 1.E.9.O	1.E.9.K.3.B = 1.E.9.Q	1.E.9.K.3.C = 1.E.9.K	
1.E.9.L.3.A = 1.E.9.N	1.E.9.L.3.B = 1.E.9.O	1.E.9.L.3.C = 1.E.9.L	
1.E.9.M.3.A = 1.E.9.P	1.E.9.M.3.B = 1.E.9.Q	1.E.9.M.3.C = 1.E.9.M	
1.E.9.N.1.A = 1.E.9.N	1.E.9.O.1.A = 1.E.9.O	1.E.9.P.1.A = 1.E.9.P	1.E.9.Q.1.A = 1.E.9.Q

Since we've expanded the game tree, there's no need to determine the value of intermediate nodes. Each party only needs to evaluate the leaves of the game tree, and then make counteroffers that drive the negotiation toward the best possible deals. The values in the following table come from the utility functions in Section 7.4

Offer	I		II	Buyer	Seller
	A	B	B		
1.A	**1**	**1**	**100:100**	**104**	**1**
1.B	**1**	**2**	**100:100**	**101**	**1**
1.C	**2**	**1**	**100:100**	**101**	**1**
1.D	**2**	**2**	**100:100**	**103**	**1**
1.E.9.N	**1**	**1**	**(100+x): (200-y)**	**3+x**	**100-y**
1.E.9.O	**2**	**1**	**(100+x): (200-y)**	**x**	**100-y**
1.E.9.P	**1**	**2**	**(100+x): (200-y)**	**x**	**100-y**
1.E.9.Q	**2**	**2**	**(100+x): (200-y)**	**2+x**	**100-y**
1.E.1.C.4	**1**	**1**	**(100+x): (200-y)**	**3+x**	**100-y**
1.E.1.C.5	**1**	**2**	**(100+x): (200-y)**	**x**	**100-y**
1.E.2.C.4	**2**	**1**	**(100+x): (200-y)**	**x**	**100-y**
1.E.2.C.5	**2**	**2**	**(100+x): (200-y)**	**2+x**	**100-y**

The following figure shows the complete game tree. The players' counteroffers are denoted by the red and blue links. Green lines connect identical states of the world, *i.e.*, transpositions.



7.8 Choosing a Strategy

Recall that the seller doesn't know the buyer's valuation, nor does the buyer know the seller's. It is clear that the seller will do better than any of the first four deals if the price exceeds \$100, *i.e.,* $y < 99$, so the first counteroffer will not include a price of \$100. The buyer is aware of this fact and assigns a low probability of reaching any of the first 4 leaves in the game tree. The seller doesn't know exactly what shoes the buyer is looking for. However, the ordering of the options in the attributes of style and color indicate that black wingtips are preferred to brown loafers. Hence, the seller assigns a higher probability of reaching leaves 1.E.9.N and 1.E.1.C.4 than either 1.E.9.Q or 1.E.2.C.5. The seller can't draw any conclusions from the initial offer, but can adjust these probabilities based on intermediate offers. For example, if the buyer counters with color=black, the seller can increase the probability of reaching 1.E.9.P and 1.E.1.C.5. This information might also prove useful in future negotiations with the same buyer.

The seller doesn't care what combination of style and color is ultimately decided upon, so the risk of a failed negotiation is minimized by keeping options open as long as possible. Thus, the seller's first counteroffer is most likely to be 1.E, namely,

style=wingtip,loafer, color=black,brown, price=200.

The buyer may misjudge this component of the seller's utility and assume that the seller would rather sell wingtips than loafers. That error may affect the buyer's judgment of the likelihood of ending up with a particular deal.

As noted in Section 6.3, the buyer can't narrow the space of style and color without precluding the possibility of getting his first or second choice. Hence, the buyer must introduce a disjunction. Recall that each disjunction must follow separately the rules of the negotiation. Hence, each disjunction is represented by a node in the game tree, so no further analysis is needed. Each disjunction will have its own valuation. At some stage, the buyer will declare one of them failed and accept the other. In our example, one disjunction will lead to 1.E.9.N, and the other to 1.E.9.Q. Since the buyer's valuation for

the former exceeds that of the latter, the buyer will declare 1.E.9.Q failed and will accept 1.E.9.N, after settling on a price, of course.

We also need to deal with the continuous variables, price in this example. We've used a special notation that indicates a specific node in the tree, 1.E.9.N, for example, is re-entered when an offer that changes just the price attribute is made. We need to assign a probability of the other party's response to each possible value of the price increment. We know, for example, that a price equal to the other party's offer is almost certain to be acceptable. We also know that a very small increment is likely to lead to a failed negotiation. Neither party knows the other's sensitivity, though. As a first guess, we might take a linear function with values between 0 and 1 as the offer varies from no price increment to agreeing with the other party's price. For example, the buyer might assign the probability that a counteroffer on price will *not* lead to a failed negotiation to be

$$P_{buyer}(p) = \frac{p-(100+x)}{100-x-y}, \quad p \le 200-y,$$

and the seller

$$P_{seller}(p) = \frac{(200-y)-p}{100-x-y}, \quad p \ge 100+x.$$

Of course, the probabilities need not be linear, or even analytic.

With the conditions specified, and both parties acting rationally, the negotiation that results can be represented by 1.E.1.C.4 with a price determined by the relative price sensitivity of the parties. Assuming both are risk neutral means that their probability of finding a price offer unacceptable is a linear function of price. Using the data from the advertisement in Section 7.3, namely that the seller wants at least $50 and the buyer won't pay more than $250, and the first offer each introduced into the negotiations, $100 and $200 for the buyer and seller, respectively, gives

$$P_{seller}(p) = \frac{p}{150} - \frac{1}{3}, \text{ and } P_{buyer}(p) = -\frac{p}{150} + \frac{5}{3}.$$

These two straight lines cross at $p = 150$, the most likely price, with a probability of being accepted of 2/3.

Of course, the two parties don't know their opponents' true limits. If the buyer thinks the seller's true minimum is $100 instead of $50, the straight lines

$$P_{seller}(p) = \frac{p}{100} - 1, \text{ and } P_{buyer}(p) = -\frac{p}{150} + \frac{5}{3}$$

intersect at $160 and a probability of 3/5. If the seller thinks the buyer's true maximum is $200 instead of $250, then the straight lines

$$P_{seller}(p) = \frac{p}{150} - \frac{1}{3}, \text{ and } P_{buyer}(p) = -\frac{p}{100} + 2$$

intersect at $140 with probability of acceptance of 3/5. It is clear that the buyer may evaluate a deal as if paying more than the true equilibrium price and the seller settling for less. Fortunately, we can use the results of the extensive literature on bargaining over one, competitive attribute for this case.

One further point needs attention. There is nothing to prevent one party from changing the other's proposal for a numerical attribute. The only rule is that the increment must be narrowed. Thus, the seller could increase the value of *x*, in essence saying. "Would you be willing to pay $10 more?" Such an offer is not guaranteeing this price, which would be done by increasing *y*. Of course, the buyer is likely to be surprised by such an offer and may declare a failed negotiation.

7.9 Introducing a New Attribute

The example in Section 7.7 did not involve introducing a new attribute during the negotiation. Had we done so, the subsequent part of the game tree would have been expanded by a factor related to the number of attribute values included. Here, we'll show the effect of such an offer.

For simplicity, we'll assume that the new attribute is introduced just before agreeing to an ultimatum. Also, for simplicity, we'll assume that the seller introduces the attribute *manufacturer* from the *shoe* vocabulary with two values Dexter and Florscheim.

Offer	I			II		IV[39]
	A	B	C	A	B	A
1.E.1.C.4.A	1	1	0	1	(100+x): (200-y)	1,2
1.E.1.C.4.A.1	1	1	0	1	(100+x): (200-y)	1
1.E.1.C.4.A.2	1	1	0	1	(100+x): (200-y)	2

As shown by this example, the new attribute can affect the buyer's and seller's valuations. It is important to carry these nodes through the game tree with the probability that they will be introduced later. Unfortunately, unless the marketplace has limited the number of legal attribute values, the probabilities assigned to the various values are likely to be unreliable.

[39] Since an attribute with an agreed upon value can never re-enter the negotiation, we've removed the delivery attribute from the table.

8 Experimental Validation

I have written a prototype that implements everything discussed thus far except for the tree search strategy. This object-oriented code includes classes related to the specification of the offers. These classes specify attributes, vocabularies, sections, contracts, and offers. The last of these enforces the negotiation protocol. The properties of the participants are captured in classes for valuations, negotiators, and a negotiation class that controls the sequence of the negotiations. This last class is the only one that assumes two-party negotiation; all the others are generic in this regard. The strategy method is a parameter passed to the constructor for the negotiator. Its valuation and strategy methods are what distinguishes it from other negotiatiors. Thus, the only thing that distinguishes a buyer from a seller is the sign of the price dependence of its valuation. While the real negotiation tool will involve sending messages between, and ultimately among, the participants, for now everything runs within a single program.

The file containing each class comes with a unit test for that class. Thus, all type checking of attribute values is done in the attribute class, and we know violations will be caught because of the tests. In addition, there are no cycles in the *has-a* relation. That means that the unit tests can also test all interactions with other classes, and a change in one class does not involve retesting any other classes.

In order to test the code developed so far, I ran a simple negotiation using the vocabularies and utility functions in Section 7. Since the strategy module isn't ready yet, I used a "random" strategy. This code steps through the attributes of the current offer. When it encounters an attribute with multiple values, it selects the first in the list as its counteroffer. If it finds an attribute with a numeric range, it selects the lower endpoint if that results in a finite valuation or the upper limit if it doesn't. A trace of the negotiation showing the changes follows.

```
(1.0, 1, [('payment', 'price', [300.0, 300.0])], 13.8)
(2.0, 0, [('payment', 'method', ['check', 'cash'])], 15.4)
(3.0, 1, [('payment', 'method', ['cash'])], 13.4)
(4.0, 0, [('shoe', 'style', ['loafer'])], 13.4)
(5.0, 1, [('shoe', 'color', ['brown'])], 13.0)
(6.0, 0, [('delivery', 'delay', [0, 0])], 12.6)
```

The first column is the time, measured in rounds. The second denotes the player. The next field is a list of changes, in this simple case only a single attribute. The final number is the player's valuation of the deal using the method described in Section 5.

I've also created a strategy that looks at the attribute values and eliminates the one with the lowest utility. I've run that strategy for the seller against a random one for the buyer with the result that follows. We see that in this simple case, the seller didn't benefit but the buyer did end up worse off. This poor behavior is a failure of the strategy to take into account the preferences listed by the buyer for the ordered attributes. When the seller doesn't care, he should base his counteroffer on his estimate of the buyer's preferences.

```
(1.0, 1, [('payment', 'price', [300.0, 300.0])], 17.8)
(2.0, 0, [('payment', 'method', ['check', 'cash'])], 15.4)
(3.0, 1, [('payment', 'method', ['cash'])], 14.4)
(4.0, 0, [('shoe', 'style', ['loafer'])], 13.4)
(5.0, 1, [('shoe', 'color', ['black'])], 16.0)
(6.0, 0, [('delivery', 'delay', [0, 0])], 8.4)
```

9 Future Work

Substantial progress has been made, but much remains to be done. The bulk of this document is a proof of concept that shows the idea of expanding the negotiation as a game tree is feasible. The infrastructure is largely complete, albeit only as a prototype. The basic mechanisms are in place, and a negotiation can be driven to completion. Several problems needed to be solved to complete the formulation. Some of these problems are straightforward applications of known ideas; others will involve solving some interesting research problems.

A real strategy needs to deal better with attributes with continuous values, such as integer and floating point value types. The difficulty of finding a value for a counteroffer has several components. First, there's the problem of finding a value that satisfies the constraints of both parties. We have an interval representing the current offer, and we know that this range overlaps the acceptable values for both parties. We don't know if this range encompasses one or both of the endpoints, is contained in one or both, or extends beyond on one side or the other. Fortunately, we have a scale factor since attributes have been programmed to disallow a doubly infinite interval.[40] Hence, we can use standard methods for finding a *feasible* solution for an optimization problem.

Once we've found a feasible solution, we need to find a new range as a counteroffer. The first step is to find the sign of the term in the utility function of either party. While it is possible to examine the utility representation, this function may be arbitrarily complex. For example, price may enter into combinations and individual terms and may have a highly non-linear dependence on time. It appears that the best approach is to treat the utility function as a black box. We can then use techniques of mathematical optimization to find a counteroffer.

Another problem that needs to be addressed is inferring the opponent's utility function. More important is coming up with a procedure to update this estimate based on the course of the negotiation. We've seen some simple ideas earlier, which need to be formalized before code implementing them can be written. We also need to convert these utility estimates into probabilities that the opponent will agree to a particular deal.

There are a number of unanswered questions. Is there a way to make use of the fact that we don't know if an attribute is cooperative or competitive for one of the parties? Can the strategy use a rank ordering, as opposed to a quantification, of the possible deals? Is the meta-negotiation that decides what sections go into a contract template identical to

[40] A doubly infinite interval would correspond to no preference. Not including the attribute has the same meaning.

the negotiation over that template? How much state must parties save to avoid cycles when restarting failed negotiations?

10 Conclusions

A search of the literature, some of which is listed in Section 11, has shown some gaps. Little work has been done that focuses on the cooperative aspects of negotiation; most of the literature centers on competitive attributes. Also, the complex offers are almost universally handled in a manner that won't scale to large numbers of attributes and attribute values. Additionally, none of the work found to date looks at the negotiation as a game to be played by expanding the game tree.

While some of the following ideas may have appeared in the literature, neither I nor the people I've asked, know of any reference to previous publication of them, at least as applied to automatic negotiation. Please let me know where these ideas have been proposed by others.

- Use of the same vocabularies for lookup and as negotiation ontologies.
- The negotiation protocol that is guaranteed to terminate, modified slightly from the protocol developed for e-speak.
- The specific valuation function used, particularly but not solely as it applies to convex hull offers.
- Finite support for attributes with continuous values.
- The particular method used for accounting for time.
- Treating the negotiation like a chess game with uncertain information of the opponent's utility.
- Dealing with continuous variables in expanding the game tree.

11 References

1. Mihal Barbuceanu and Wai-Kau Lo, "A Multi-Attribute Utility Theoretic Negotiation Architecture for Electronic Commerce", presented at Agents 2000, Barcelona, Spain, (2000)
2. R. Lewicki, D. Saunders, and J. Minton, *Essentials of Negotiation*, Irwin (1997)
3. P. Faratin, C. Sierra, and N. R. Jennings, "Using Similarity Criteria to Make Negotiation Trade-offs", *International Conference on Multiagent Systems (ICMAS-2000), Boston, MA.* pp. 119-126 (2000) (describes trade-offs on indifference curve that improves opponent's utility; ideas on estimating opponent's utility function and "distance" between deals)
4. N. R. Jennings, S. Parsons, C. Sierra, and P. Faratin , "Automated Negotiation" Proc. 5th Int. Conf. on the Practical Application of Intelligent Agents and Multi-Agent Systems (PAAM-2000), Manchester, UK, pp. 23-30 (2000) (refers to an offer being a "point or a region" in the agreement space.
5. P. Faratin, C. Sierra and N. Jennings Negotiation Decision Functions for Autonomous Agents in *Int. Journal of Robotics and Autonomous Systems*, 24(3-4):159-182 (1997)
6. H. Raiffa, *The Art and Science of Negotiation*, Harvard University Press (1982)

7. http://www.nara.gov/genealogy/soundex/soundex.html
8. R. L. Keeney and H. Raiffa, *Decisions with Multiple Objectives: Preferences and Value Tradeoffs*, Cambridge University Press (1993)
9. "Conceptual Model for Electronic Contract Framework in B2B Systems", Morciniec, Michal; Bartolini, Claudio; Boulmakoul, Abdel, HP Labs Technical Report HPL-2001-10, http://lib.hpl.hp.com/techpubs/2001/HPL-2001-10.pdf (2001)