A Multiagent Framework for Automated Online Bargaining

Fu-ren Lin and Kuang-yi Chang, National Sun Yat-sen University

The rapid development of Internet technologies is making online shopping and business-to-business purchasing an increasingly attractive option. For example, search engines and electronic catalogs are useful for researching products and vendors. More advanced still are automated bargaining agents—such as eMediator, AuctionBot, Tete-a-Tete, and Kasbah—which facilitate price negotiations between buyers and sellers.\(^1\)\(^2\) Research shows that people might actually prefer shopping online at e-stores that offer the chance to negotiate prices, even if they don’t end up with the lowest price.\(^3\) Despite this clear opportunity, few electronic stores and online business-to-business operations currently offer automated bargaining.

There are several reasons for this. First, the market determines pricing for products currently sold at e-stores, and sellers see no benefit to bargaining. Second, e-stores that let buyers bargain without automated support for the process are likely to be overwhelmed by numerous simultaneous transactions. Finally, existing automated support lacks the intelligence required to handle real-world situations. To meet demands, software agents must be capable of learning from experiences. Also, existing bargaining agents use simple, fixed strategies to issue prices and cannot handle complex situations.

To address this, we’re developing a multiagent framework to support a dynamic, online bargaining process. Our online dynamic bargaining (ODB) system uses three agents: one that uses data mining techniques to generalize bargaining patterns, one that matches the current transaction to an existing pattern to facilitate the best vendor response, and a third that dynamically issues prices based on utility theory.

We’ve prototyped the ODB system as a bargaining system for various applications, such as online used-car trading. We also use it as a reference in our ongoing development of intelligent bargaining systems, comparing it against different intelligent bargaining approaches, including neural networks and persuasive mechanisms. Here, we describe how the ODB system works and show results from a field experiment in which we compared its performance to that of a similar system in terms of the seller’s profit and customer satisfaction.

**ODB system framework**

A bargaining process consists of a series of price shifts by both sellers and buyers. A deal is made when both sides agree on a proposed price. An automated bargaining system supports this process first by issuing prices according to updates from both sides, then by deciding to accept an offered price.

The ODB system works on the seller side using a multiagent framework to bargain prices with buyers. Figure 1 shows the ODB system architecture. The system consists of three agents: a dynamic price-issuing agent, a pattern generalization agent, and a pattern-matching agent. The dynamic price-issuing and pattern-matching agents execute in front-end settings, interacting with buyers in real time. The pattern generalization agent executes in back-end settings, processing transaction data periodically to generate bargaining patterns in batch mode.

We define the bargaining pattern as follows: The seller’s price series is denoted as \( S = \{ S_0, S_1, S_2, ..., S_n \} \) and the buyer’s is denoted as \( B = \{ B_0, B_1, B_2, ..., B_n \} \). \( S_i \) and \( B_i \) indicate prices issued at the \( i \)th step by a seller and a buyer, respectively. Figure 2 shows an example of a bargaining process, including gain and premium, which we describe in more detail later. A bargaining pattern generalizes both price series (\( S \) and \( B \)) and

\[ \text{Gain} = S_i - B_i \]

\[ \text{Premium} = S_i - S_{i-1} \]

This framework allows for dynamic price adjustments based on the current transaction and previous interactions, leading to more efficient and satisfactory outcomes for both parties.
Assume that there are \( N \) transactions (\( N = n \)), and each transaction goes through \( m \) bargaining states (for example, \( S_0, B_0, S_1, ...; S_{m-2}, B_{m-1} \text{, and } P_{\text{end}} \)).

1. Create \( m \times N \) matrices and fill \( e \) in each matrix;
2. Calculate \( E \) for each fragment and generate \( m = 2 \) \( E \)-matrices;
3. Check all cells in \( E \)-matrices and remark the cells that exceed \( \sigma \) with \( \infty \);
4. Sum up \( m = 2 \times N \times E \)-matrices and generate \( m \times N \) \( TE \)-matrix; \( //TE = E_1 + E_2 + ... + E_{m-2} \);
5. Do {
   Select a cell of \( TE \)-matrix having the minimum \( TE \) value;
   Generate the pattern with \( m \) bargaining states;
   \( N = n - 1 \);
   Update \( m = 2 \times N \times E \)-matrices and \( N \times N \) \( TE \)-matrix with the new pattern;
   ) Until no more new patterns can be merged;

### Learning phase: Bargaining pattern generalization

In the learning phase, the pattern generalization agent generates bargaining patterns represented by the price-shifting slopes on both seller and buyer sides. After a deal is made, the ODB system stores the bargaining process, including the series of seller and buyer prices, in the database. In a batch fashion, the pattern generalization agent summarizes these price changes, generates the common price-shifting slopes, and then stores them in the pattern base.

Based on this historical transaction data, the pattern generalization agent discovers bargaining patterns. With sequential pattern algorithms, viewing a bargaining process as a sequence of discrete states makes it easier for the algorithms to generate bargaining patterns effectively. However, the bargaining pattern represents the common price-issuing sequence from the beginning to the end of the process, whereas the sequential pattern algorithms find the common parts from different time stamps. Given this, we developed a pattern generalization approach by clustering similar price series fragment by fragment, as the pseudo code in Figure 3 shows.

After transactions, the system asks the system administrator to evaluate the performance of the bargaining outcomes in terms of the seller’s gain and buyer’s satisfaction. The pattern generalization agent uses the transaction data classified as successful deals for updating bargaining patterns.

We generalize common bargaining patterns in four steps.

In Step 1, we fragment the price-issuing series. We define a fragment of a slope list of buying prices as an \( SoB \) segment with three contiguous buying prices. We calculate the distance value of two corresponding \( SoB \) fragments each time. Each \( e \) represents the difference between the two corresponding \( SoB \) buying prices, and \( E_e \) is equal to \( e_1 + e_2 + ... + e_e \). \( \sigma \) is set as a threshold of \( E_e \) and if \( E_e \) is greater than \( \sigma \), two \( SoB \)s are seen as two different bargaining patterns. For example, let \( SoB_1 = \{A_{10}, A_{11}, A_{12}, ... , A_{1n}\} \) and \( SoB_2 = \{A_{20}, A_{21}, A_{22}, ... , A_{2n}\} \) denote two slope lists of buying prices. Given this, we get \( E_1 = (A_{10} - A_{0}) + (A_{11} - A_{1}) + (A_{12} - A_{2}) \), \( E_2 = (A_{20} - A_{0}) + (A_{21} - A_{1}) + (A_{22} - A_{2}) \), and \( E_{0} = (A_{10} - A_{0}) + (A_{11} - A_{1}) + (A_{12} - A_{2}) \). If any \( E_{0} \) exceeds \( \sigma \), we assert that these two series are different. We apply the same fragmentation process to \( SoS \).

In Step 2, we compare two corresponding price bargaining states. If no \( E_p \) in two \( SoS \)s exceed the threshold, we view the two buying price series as identical. Moreover, we must ensure that the \( E \) values of two corresponding \( SoS \)s are also identical. When no \( E_p \) of the two corresponding \( SoS \)s exceed the
threshold, the two SoBs and SoSs generate the bargaining pattern.

In Step 3, we generate bargaining patterns. If two corresponding SoBs differ within σ, we generate a bargaining pattern by replacing the slopes of the two SoBs with the means of the corresponding slope values. For example, Z denotes the two buyers’ price-issuing pattern, and the slope series of Z is Z₀, Z₁, Z₂, and Z₃, where \( Z₀ = \frac{(A₀₁ + A₀₂)}{2}, Z₁ = \frac{(A₁₁ + A₁₂)}{2}, Z₂ = \frac{(A₁₂ + A₁₃)}{2}, \) and \( Z₃ = \frac{(A₁₃ + A₁₄)}{2} \). We generate the slope list for the corresponding selling-price pattern in the same way. Both slope lists comprise the bargaining pattern and are stored in the pattern base for matching future transactions.

In Step 4, we repeat Steps 2 and 3 to check all price series that buyers and sellers have issued until we generate no more new patterns.

How do we determine the threshold of \( E \) denoted by \( σ \)? If we set \( σ \) at a high value, fewer patterns are generalized, which results in a larger pattern-matching hit rate. On the other hand, if we set \( σ \) at a low value, we generate many patterns and a smaller hit rate.

Given this, we determine \( σ \) in several steps. First, we determine the price difference that the seller would view as insignificant. For example, say two buyers offer $8,400, and then increase it to $8,500 and $8,600, respectively. If a seller sees no significant difference between $8,500 and $8,600, then the $100 difference between them would be the insignificant price difference.

Next, based on the concept of normalization, we obtain \( σ \) by dividing the price difference by \( S₀ \)—in this case, 100 / \( S₀ \). Therefore, the first buyer’s slope is \( (8,500 – 8,400) / S₀ \), and the second buyer’s slope is \( (8,600 – 8,400) / S₀ \). By subtracting the first slope from the second slope, we get 100 / \( S₀ \).

**Execution phase**

During the execution phase, the bargaining process begins when individual buyers submit prices to the ODB system through a Web interface. This initiates two processes. First, the pattern-matching agent retrieves bargaining patterns that match patterns in the patterns database to determine which price to accept. If no patterns match, the system invokes the dynamic price-issuing agent to issue a price that accords with the price-issuing algorithm.

**Pattern matching.** To match different price scales in different product categories, our matching algorithm must solve price and time scaling problems. To normalize price differences among product categories, we use the percentage of price change. There are several existing techniques to normalize time scale, including using dynamic time-warping techniques, semi-Markov and segmental Markov models, or simply interpolating points to obtain symmetric curve points. In our current implementation, we adopt the interpolation approach. The pseudo code for our matching algorithm is shown in Figure 4.

The ODB system is bargaining with a buyer. If the buyer does not accept the issued price, the bargaining matching agent will return its price according to three steps.

- **Pattern match and selection.** When a buyer starts the bargaining process, he or she will read the price \( S₀ \) that the ODB system issues. When the buyer returns the price \( B₀ \), the matching agent can calculate \( A₀ \) of SoB immediately (that is, \( A₀ = (B₀ – S₀) / S₀ \)). Let \( σ \) be the threshold of \( E \). According to \( A₀ \), the matching agent calculates \( E \) based on patterns stored in the pattern base.

  - **Issuing price.** The matching agent issues the price according to the sales price series of the matched bargaining pattern.

  - **Continuous pattern matching and price issuing.** After the buyer offers a price, the matching agent can obtain the buyer’s price series (for example, \( A₁ \) and \( A₂ \)). The matching agent then checks the \( E \) value (for example, \( e₁ + e₂ \)), and if the \( E \) value is less than \( σ \) (for example, \( e₁ + e₂ < σ \)), the matching agent performs Step 2 in the algorithm in Figure 4. If \( E \) is equal to or greater than \( σ \) (for example, \( e₁ + e₂ ≥ σ \)), the matching agent searches the alternative pattern with the minimum \( E \) and uses it as the bargaining pattern, and then goes to Step 2. When the matching agent fails to allocate any bargaining patterns, the ODB system invokes the dynamic price-issuing agent to take over the online bargaining process.

**Dynamic price issuing.** When we use curve matching to identify bargaining strategies, the system sometimes fails to find a matched pattern because none exists. This can also happen when buyers’ significantly alter their bargaining behaviors. To deal with this, we use a dynamic price-issuing method to issue new prices. The process involves several steps.

  1. First, we set up the expected target price range. To determine the range, we either use historical transaction data or—if little or no historical data exists—use cost and expected profit figures. For example: Let \( C \) be the cost of an item, and \( ε \) the expected profit. The ODB system makes the deal when the price is higher than \((C + ε)\) or between \(C\) and \((C + ε)\). The acceptance region is the price equal to or greater than \(C\). During the bargaining process, the ODB system first determines if a buyer’s price is greater than \((C + ε)\). If it is, the deal is made. If it’s not, the system waits for the price to settle down between \(C\) and \((C + ε)\)—ideally, as close to \((C + ε)\) as possible.

  2. Second, we initiate the bargaining process. We built the dynamic price-issuing algorithm to include concepts of gain and premium. The system considers gain and premium only after the user’s “opponent” offers a price. That is, when the buyer issues a price, the seller con-

![Figure 4. The pseudo code for our matching algorithm.](image-url)
1. If related historical prices exist
   Then set $P_{\text{end}} = \text{the upper bound}
   Else set the $P_{\text{end}} = C + \varepsilon$; // $C$ is the cast and $\varepsilon$ is the expected profit.
2. While (deal does not close)
   Get the price $B$ that the buyer issued;
   Calculate the premium and the gain;
   If (premium / gain) < $K$ and $B > C$
   Then close the deal and exit
   Else (Calculate elicitation;
   Calculate the new price $S$ using seller’s utility;
   If $C \leq S < P_{\text{end}}$ then issue $P_{\text{end}}$;
   If $S < C$ then stop the bargaining process;
   Else issue $S$;

Figure 5. The algorithm for our dynamic price-issuing agent.

<table>
<thead>
<tr>
<th>System</th>
<th>Cost</th>
<th>$C + \varepsilon$</th>
<th>$S_{0}$</th>
<th>Elicitation Threshold</th>
<th>$\sigma$</th>
</tr>
</thead>
<tbody>
<tr>
<td>ODB system</td>
<td>$5,300</td>
<td>$7,700</td>
<td>$9,500</td>
<td>0.05</td>
<td>0.021</td>
</tr>
<tr>
<td>LD system</td>
<td>$5,300</td>
<td>N/A</td>
<td>$9,500</td>
<td>*</td>
<td>N/A</td>
</tr>
</tbody>
</table>

* A seller will issue a new lower price after a buyer insists on the same price three times in average using a random function.

siders the premium and gain, but the buyer does not consider them until the seller issues a counter offer. $U_{b}(x_n)$ denotes the seller’s utility function, where $x_n = B_n - B_0$ and $n = 1, 2, \ldots, t$. $U_{b}(y_n)$ denotes the buyer’s utility function, where $y_n = S_n - S_0$ and $n = 1, 2, \ldots, t$. The seller’s gain when the buyer issues $B_n$ is $U_{b}(B_n - B_0)$, and the buyer’s gain when the seller issues $S_n$ is $U_{b}(S_n - S_0)$. The buyer’s premium is defined as the current sales price minus the buyer’s ideal price (that is, $U_{b}(S_n - P_{\text{end}})$). The buyer’s premium is defined as the buyer’s idea price minus the current buying price (that is, $U_{b}(P_{\text{end}} - B)$).

In the final step, the price-issuing agent reaches an acceptance decision based on a consideration of gain to premium. Figure 2 shows this process. At time $t_4$, the buyer issues a price ($B_4$), the dynamic price-issuing agent then either accepts the buyer’s price or issues another price. In this situation, the seller gains $U_{b}(B_4 - B_0)$ and loses premium $U_{b}(P_{\text{end}} - B_4)$. The dynamic price-issuing agent accepts the buyer’s price if the gain is equal to or greater than the premium and the price is located within the acceptance region.

If a buyer is a risk-taker and prefers not to pay the premium, the seller might allocate the $P_{\text{end}}$ higher than the ideal $P_{\text{end}}$ to obtain a higher profit. On the other hand, if a buyer is risk averse, he or she will accept the price issued by the seller when the gain equals the premium. Because a seller cannot immediately plot the buyer’s utility functions, the seller cannot decide how much to lower the sales price to increase a unit of the buyer’s utility. However, a seller can decide the selling price for decreasing one unit of his or her own utility. With this method, the price-issuing algorithm determines the price-dropping range based on the seller’s utility function.

When the ODB system receives a buyer’s price that is below the selling price, it does not always lower the asking price. Even if a buyer insists on his or her issuing price, the seller still must estimate carefully before lowering the price. We denote the seller’s tendency to willingly reduce a sales price as elicitation. The higher the elicitation, the more easily the agent tends to issue a lower price. In the dynamic price-issuing algorithm, we use the concept of utility to deal with elicitation. For example, a buyer issues $8,200$ at $t_1$, and issues $8,300$ at $t_2$. With $U_{b}(s)$ denoting the seller’s utility function, the elicitation will be $U_{b}($8,300$) - U_{b}($8,200$). As the buyer increases the buying price, the seller’s elicitation increases. When the elicitation exceeds the threshold, the seller will issue the lower price. Formally, the elicitation threshold ($ET$) is defined as the minimum increase of a seller’s utility—the point at which the seller will lower a sales price in relation to the buyer’s price range increase.

We present the algorithm for the dynamic price-issuing agent in Figure 5.

Table 1 shows the basic settings for the experiment. We set UNS at 0.025. For UDS, we started at 0.025 and decreased it 20 percent each step until it reached 0.008, and set UIS at 0.008, increasing it 20 percent each step until it reached 0.025.

To answer these questions, we conducted a field experiment to evaluate and compare the ODB system with another online dynamic bargaining system—Liang and Doong’s system. We chose the LD prototyping system as the benchmark because both systems have similar environments and performance measures, and they both take the seller’s risk perspective in issuing prices. The ODB system differs from the LD system in that it has learning and elicitation handling capability.

Based on the sellers’ risk perspectives, we evaluated the systems in three bargaining settings:

- risk seeking, which corresponds to the LD system’s utility decreasing strategy (UDS);
- risk neutral, which corresponds to the LD system’s utility increasing strategy (UIS);
- risk averse, which corresponds to the LD system’s utility neutral strategy (UNS).

Table 1 shows the basic settings for the experiment. We set UNS at 0.025. For UDS, we started at 0.025 and decreased it 20 percent each step until it reached 0.008, and set UIS at 0.008, increasing it 20 percent each step until it reached 0.025.

To attract participants, we posted call-for-subjects messages on major bulletin board systems and commercial Web sites. When subjects first enrolled, we asked them to fill out a personal profile that included gender, education, and age. During our seven-day experiment, 207 subjects joined and we collected 199 transactions (a few transactions were incomplete). Most subjects were male (89.05 percent), ranged in age from 15 to 30 years.
years old (91.95 percent), and were well educated (88.95 percent, including university and graduate degrees).

Our experiment consisted of three main stages: introduction, experimentation, and a questionnaire. When users logged into the experimental system, they read a Web page that introduced the experiment, which included a discussion of the experimental procedure and purpose and information on providing feedback. After reading the introduction, users could proceed with the experiment.

During the experiment, all users (buyers) had the same system interface and were not told which bargaining system they were using. Because it is popular and has a relatively wide price range, we selected Ericsson’s T28sc cellular phone as the product for bargaining. Selling only one product perhaps made our electronic store look a little shabby, but it avoided our having to deal with things like different product features. After closing the deal, we asked buyers to fill out a questionnaire, which we used to measure customer satisfaction.

**Bargaining systems**

We used two measures to evaluate the two bargaining systems:

- the seller’s financial gain, which we defined as the profit obtained from bargaining transactions, and
- the buyer’s satisfaction with the system.

\[ P_i = \sum_{i=1}^{n} (P_i - C) \]

and \( n \) is the number of deals. We measure a buyer’s satisfaction based on responses to our online survey. Figure 6 shows the questionnaire (translated from Chinese).

We were also interested in understanding how the ODB system works—particularly, how the three agents would interact in the bargaining process across contiguous periods.

In the experiment, we divided the two bargaining systems into six bargaining settings that were randomly selected to interact with incoming buyers. At the end of each of the seven days, the generalization agent generalized bargaining patterns by collecting data from the first day to the current day. The system then used the generalized patterns to match transactions the following day. We selected high seller’s gain transactions for the generalization agent to generalize good patterns, and then we loaded patterns into the ODB system at the start of each day. If the patterns were useful, the seller’s average daily gains should grow smoothly each day. We used the hit rate, defined as the percentage of deals matching the bargaining patterns, to evaluate the ODB system’s adaptability. We expect the hit rate to grow as the bargaining process proceeds over time.

**Results**

Given the bargaining pattern’s incremental refinement, we can best understand the ODB system by analyzing the bargaining pattern’s hit rate and the fluctuation in the seller’s gain over the test period.

**Partial hits**

Partial hits are the number of matches over the course of a transaction that have a different end price than the matched pattern end price within \( \sigma \). We use the partial hit rate to calculate the ratio between partial hits and transaction length. We define the hit-occurring rate as the total percentage of transactions with partial hits in a given day. Finally, we define hit rate as the hit-occurring rate multiplied by the mean of the partial hit rate. Figure 7 shows the growth of hit rate over seven days. As expected, the hit rates increase over time. The exception to this trend was that on Day 5, the hit rate was lower than Day 4. The reason for this was that users that day created a lot of noise: We found many negative, extraordinarily large, or meaningless numbers from carelessly issued prices.

We view the evolution of the ODB system using the average gain in each day as shown in Figure 8a.

**Seller’s gain**

First, we conducted \( t \)-tests to compare the seller’s gain between Day 1 and Day 7 in risk seeking, neutral, and averse attitudes, respectively. The results of the \( t \)-tests show that the seller’s financial gain significantly increases over the period if we set the ODB system to risk seeking or risk neutral. In fact, even the risk-averse setting increases financial gain, but only by a little. This might be because the risk-averse setting always gets lower profit.

**Figure 6. The customer satisfaction survey. For each question, we used a five-point Likert scale, ranging from 1 (strongly disagree) to 5 (strongly agree).**
than the other two settings, and thus they cannot be the source of generalized patterns. Given the results, we conclude that the ODB system can increase the financial gain after a period of evolution. Comparing hit rate with the gain fluctuation, we found a positive correlation between hit rate and gain. This implies that the ODB system’s adaptability is capable of improving its bargaining strategy based on past experiences.

Next, we compared the two systems’ average gain. As Figure 8a shows, UDS gradually offered small discounts, and a buyer will feel utility decrease as the bargaining process goes on. Indeed, as the graphs in Figure 8 show, the average seller’s gain from the LD system fluctuated, while the ODB system exhibited stable performance in every setting.

To compare average gains of each system we used a t-test. The results show that the ODB system with risk seeking and neural settings offers significantly greater gains than the LD system with UDS and UNS strategies. The ODB system with the risk aversion setting did not significantly outperform UIS strategy. From these results, we conclude that, in most situations, the ODB system creates higher profit than the LD system.

**Customer satisfaction**

We next evaluated customer satisfaction using the experimental systems. We estimated the questionnaire’s reliability using Cronbach α and its value is 0.9273. We can therefore assert that the satisfaction questionnaire is reliable.6

Because each question bears on the final question, we can correlate between them. Given that our results show that all questions are significantly correlated, we estimate customer satisfaction based on the final question.

The average satisfaction rating for the ODB system was 3.47 and that for the LD system was 3.42. With a medium score of 3.00, both systems are significantly higher than 3.00 at the 0.05 level (two-tailed t-test). However, neither scored significantly different at the 0.05 level. Therefore, we assert that customer satisfaction with the two systems was relatively equal.

Our work with the ODB system initiates research on generalizing bargaining patterns to facilitate automated online price bargaining for e-trading. Along with this contribution, we obtained better algorithms in automated, intelligent price

---

**Figure 7.** The ODB system’s hit rates during the seven-day field experiment.

**Figure 8.** The average gain comparison between the ODB system and the LD system using three different settings. We compared the ODB system’s (a) risk seeking with the LD system’s UDS, (b) risk neutral with UNS, and (c) risk averse with UIS. Gain is shown in Taiwanese dollars (1 US dollar = 35 Taiwan dollars).
There are several existing, experimental automated bargaining systems—including AuctionBot, Kasbah, Tete-a-Tete, and eMediator—that use multiagent techniques. AuctionBot is a general purpose Internet auction server that lets users launch new auctions by choosing from different auction types with specified parameters. Buyers and sellers can then bid according to the selected auction's multilateral distributive negotiation protocols. In the AuctionBot marketplace, a seller initiates the auction after selecting a reserved price, then lets AuctionBot manage the bidding process according to the auction protocol and parameters. Kasbah is a Web-based multiagent system that uses both buying and selling software agents to help users in the marketplace. Kasbah agents do not use any artificial intelligence or machine-learning techniques. In Kasbah, when a seller creates a new selling agent, he or she sets up several guiding parameters. These parameters include the date by which you must sell the item, the desired price, and the lowest acceptable price. Kasbah has a bargaining strategy function that guides agents in issuing prices in the bargaining process. The system has three basic price-issuing functions: linear, quadratic, and cubic, which represent anxious, cool-headed, and greedy or frugal bargaining attitudes, respectively.

Tete-a-Tete's buying and selling agents negotiate cooperatively in transactions across multiple terms, including price, warranties, delivery times, service contracts, return policies, loan options, gift service, and other value-added merchant services. Tete-a-Tete's buying agents negotiate argumentatively with selling agents, and use the evaluation constraints captured during product selection and merchant brokering stages as dimensions of a multiattribute utility.

eMediator is a next-generation electronic commerce server enhanced by AI and algorithmic techniques, game-theory incentive engineering, and GUI design. eMediator has two agents: eAuctionHouse, a configurable auction house with various parameterizable auction types, and eCommitter, a levelled commitment contract optimizer.

Liang and Doong’s electronic mall prototype experiments with identifying how bargaining attracts customers, which bargaining strategies are effective, and how individual difference impact bargaining outcomes. The electronic mall is composed of several stores, some of which offer bargaining. They use three bargaining strategies:

- the utility decreasing strategy, which makes a large discount on the initial offer, followed by smaller and smaller concessions, so that buyers perceive decreasing utility;
- utility increasing strategy, which makes a small discount on the initial offer, followed by increasingly large concessions, so that buyers perceive increasing utility; and
- utility neutral strategy, which uniformly discounts the price so that buyers perceive constant utility.

References


The Authors

Fu-ren Lin is an associate professor in the Department of Information Management, National Sun Yat-sen University, Taiwan. He received his MS in computer science and information engineering from the National Chung-cheng University, Taiwan, and his PhD in information systems from the University of Illinois at Urbana-Champaign. His research interests are business process reengineering, data mining and knowledge management, and electronic commerce. Contact him at the Department of Information Management, National Sun Yat-sen University, Kaohsiung City 804, Taiwan; frlin@cc.nsysu.edu.tw.

Kuang-yi Chang is fulfilling his military service after receiving his MS from the Department of Information Management National Sun Yat-sen University. His research interests are in electronic commerce and data mining. Contact him at master1@ms9.hinet.net.