An Asynchronous Negotiation Framework Based on Answer Set Program for E-commerce

Baobao LI, Liang LI, Yiming HE, Wu CHEN*

College of Computer and Information Science, Southwest University, Chongqing 400715, China

Abstract

Electronic commerce has been a significant commercial phenomenon for many years, and automatic bargain between buyers and sellers further increases the advantage of the e-market. However, in some scenarios of shopping online, commodity prices can no longer be the core of negotiation, for price may be always unchanged in the negotiation process. This paper proposes a logic framework for asynchronous negotiation and update based on Answer Set Program (ASP), in which the above negotiation scenario in electronic commerce is distinguishingly discussed, and a method for the seller to make a decision on price is proposed.

Keywords: Negotiation; Answer Set Program; E-commerce

1 Introduction

Nowadays, shopping online is indispensable in our daily life, negotiation as a effective way to help the buyers and the sellers to reach an satisfying agreement has been researched in the fields of game theory and artificial intelligence for many years [1-4]. In the fields of artificial intelligence, the methodology of logic framework for negotiation has helped us to established a qualitative methodology for reasoning about bargaining and negotiation, which differentiates them from the traditional game-theoretic approaches to bargaining problems [5], for the traditional game-theoretic approaches are often considered as a quantitative methodology. The traditional game-theoretic approaches focus on commodity prices, buyers and sellers bargain on the commodity price. However, in the negotiation scenario of electronic commerce, in order to improve efficiency of trading, some sellers are not to compromise on the price, instead, they provide some other benefits for the buyers. Aiming at this special scenario, a feasible logic-based method to resolve the negotiation problem of this special scenario is presented in this paper. Answer Set Program (ASP) as a non-monotonic reasoning declarative logic program has been applied to negotiation widely for its rationality. In this paper we model an agent’s knowledge base as an ASP, answer sets of an ASP represent the agent’s negotiation demands, and negotiation between agents becomes the modification process of their demands and update of their ASPs, hence the negotiation between

*Corresponding author.
Email address: chenwu@swu.edu.cn (Wu CHEN).
a seller and many buyers becomes negotiation between the seller’ ASP and the buyers’ ASP. The seller synchronously negotiate with many buyers, but each sub-negotiation among the seller and every buyers is independent, this can be explained by every buyer doesn’t want his private matters to be known by other buyers.

In the process of each sub-negotiation, the buyers updates their ASPs while the seller doesn’t. Given each sub-negotiation terminates in different time, and new sub-negotiation may be added, the update of the ASP of the seller must be asynchronous, i.e. the seller updates his ASP after negotiate with a certain number of buyers, but this update must synthesizes the results of each sub-negotiation. This update process can help the seller to make a decision on this goods. For example, according to the previous transaction records a seller can decide whether to raise prices or cut price, or just keep the price staying the same.

2 Preliminaries

An ASP defined over a propositional language consists of rules of the form

\[ L_0 \leftarrow L_1, \ldots, L_m, \text{not} L_{m+1}, \ldots, \text{not} L_n \]

Where \( n \geq m \), \( L_i \) is literal, i.e., either propositional atom or its negative form. The symbol \text{not} denotes negation by failure (weakly negation), \( \neg \) denotes classical negation (strong negation). The left-hand side of a rule \( r \) is called the head of \( r \) and the right-hand side is the body of \( r \), respectively denoted by \text{head}(r) \text{ and body}(r) \text{, } Pos(r) \text{ denotes } (L_1, \ldots, L_m) \text{, } Neg(r) \text{ denotes } (l_{m+1}, \ldots, L_n). \text{ Then we can use the following form to denotes a rule: }

\[ \text{Head} \leftarrow \text{Pos}, \text{notNeg}, \]

or in an even simpler fashion:

\[ \text{Head} \leftarrow \text{Body} \]

Given an ASP \( \Pi \), we denoted \text{Head}(\Pi) = \bigcup_{r \in \Pi} \text{Head}(r), \text{ Pos}(\Pi) = \bigcup_{r \in \Pi} \text{Pos}(r), \text{ Neg}(\Pi) = \bigcup_{r \in \Pi} \text{Neg}(r) \text{. A rule with empty body is a fact and a rule with empty head is a constraint, hereby, we can use Fact(\Pi) to denote the set of all facts for an ASP } \Pi \text{. We say } L \text{ and } \neg L \text{ are complementary literals, i.e. they are conflicting literals. A set } S \text{ is inconsistent if it contains a pair of complementary literals. Otherwise, } S \text{ is consistent. An ASP is consistent if it hasn’t inconsistent answer set. ASP are useful for its expressive power in knowledge representation, moreover they allow a programmer directly to represent incomplete knowledge about the world [6]. And now there already exists semantic defined for extended logic programs, such as answer set semantics. Now let us introduce answer set. We say a rule } r \text{ of the form above is defeated by a literal } L \text{ if } L = L_i \text{ for some } i \in \{m + 1, \ldots, n\}. \text{ We say } r \text{ is defeated by a set of literals } X \text{ if } X \text{ contains a literal that defeats } r \text{. }

\textbf{Definition 1} \text{ Let } \Pi \text{ be a logic program, } X \text{ a set of literals. Then the reduct, } \Pi^X, \text{ of a program } \Pi \text{ relative to a set } X \text{ of atoms is defined by }

\[ \Pi^X = \{ \text{Head}(r) \leftarrow \text{Pos}(r) \mid r \in \Pi, \text{Neg}(r) \cap X = \emptyset \}. \]

An operator \( \gamma_P \) from sets of literals to sets of literals is defined as follows:

\[ \gamma_P(X) = \text{Cn}(\Pi^X) \]
$X$ is an answer set of $\Pi$ iff $X = \gamma_P(X)$.

A literal $L$ is a consequence of a program $P$ under answer set semantics, denoted $P \models L$, iff $L$ is contained in all answer sets of $P$. For more about answer set semantics, we can refer to [7, 8].

3 Asynchronous Negotiation Framework for E-commerce

3.1 The representation of knowledge base and negotiation demands

This paper we will present an asynchronous logic-based negotiation framework, in which negotiation knowledge base is represented by ASP and every answer set of a ASP is a candidate of negotiation demands of the agent. Next, we will illustrate how to represent an agent’s knowledge base and demands with an ASP and its answer sets by means of analysing an example, which is already proposed in the paper of Li Liang et al. [9].

Example G1: A staff wants to get a high salary, he knows that in order to get high wages, he must work in right ways, and he also knows that no pains no gains. The staff has already mastered the correct way to work, and make up his mind to work hard. Then we will signify the description in natural language as follows: Want to get high salary: $\text{High\_wage}$; Staffs work very hard: $\text{hard\_working}$; To master the correct working method: $\text{right\_way}$; Can’t get high salary: $\neg\text{High\_wage}$. Then we can have a extended logic program $\Pi$ according to the above example as follows:

$$
\text{Right\_way} \leftarrow \\
\text{Hard\_working} \leftarrow \\
\text{High\_wage} \leftarrow \text{Hard\_working}, \text{Right\_way} \\
\neg\text{High\_wage} \leftarrow \neg\text{Hard\_working}
$$

$\text{ANS}(\Pi) = \{\text{High\_wage, Hard\_working, Right\_way}\}$, as this program only have an answer set, it surely goes without saying that the demands of the staff can be represented by $\{\text{High\_wage, Hard\_working, Right\_way}\}$. When a program has more than one answer set, we have to select a preferable answer set as the agent’s negotiation demands. In this work, how to choose a preferable answer set as the agent’s negotiation is not our point, we will simply denote a agent’s negotiation demands as $\text{ANS}(\Pi)$ wrt.its negotiation program $\Pi$.

In the above example, literals such as $\text{Hard\_working, Right\_way}$ in the answer set are just facts of the staff’s negotiation program, not his demands negotiated with his boss, however, considering facts as negotiation demands doesn’t affect final negotiated outcome.

3.2 The negotiation scenarios

In the process of normal bilateral negotiation, in order to reach an agreement, each agent must modify his demands and ASP, and this modify process should be synchronized with negotiation. However, in some one-to-many negotiation scenarios, the modify of some agent’s demands and ASP can be happened at the end of negotiation process, i.e., this update process is asynchronous. These one-to-many negotiation scenarios can be found in the aspect of e-commerce such as a negotiation between an online shopping mall with its purchasers, or even a negotiation between
a company and its employees. Difference between them is that the number of employees may be
fixed, while the number of purchasers is constantly changing.

In the above negotiation scenario between a company and its employees, the employees all have
their own negotiation demands, these negotiation demands are not only individual privacy that
may not be informed by other employees, but also demands among employees may conflict, so the
company must synthesize every employees’ demands and meet with demands of the vast majority,
rather than the minority. For these reason, negotiation among the company and its employees
must be independent or one by one, and the company updates its demands and ASP after all
negotiations with each employee are over. With regard to negotiation between an online shopping
mall with its buyers, on account of the unceasing addition of new users, updating demands and
ASP of the online shopping mall can be at regular intervals, in these intervals, the mall in turn
or simultaneously negotiates with his users who have items to negotiate, and records demands of
his users for the subsequent updating of his ASP and demands. The focus of our work in this
paper is on the negotiation scenario where one seller and two or more buyers negotiate for single
product.

3.3 The process of negotiation

In the negotiation process of our work, the seller’s knowledge base is divided into two parts, one
is negotiation knowledge base on a commodity, the other is his decision-making knowledge base,
respectively denoted by ASPs $\Pi_k$ and $\Pi_i$. The buyers’ knowledge base is denoted by $\Pi_i$, where
$i > 0$ represents the count of buyer. Then the sub-negotiation and the negotiation are denoted by
$N_i(\Pi_k, \Pi_i)$ and $\sum_i N(\Pi_k, \Pi_i)$. For the seller, his negotiation knowledge base $\Pi_k$ is open for
his buyers, and his decision-making knowledge base $\Pi_i$ be kept in secret for the buyers. In the
process of the whole negotiation, the buyer’s negotiation knowledge base $\Pi_k$ is unaltered, i.e., the
seller only concerns about the update of his ASP at some point-in-time in the future.

As it is stated in the paper of Chen et al. [10], negotiation is a process of giving up parts of own
demands and accepting the opponents’ demands, i.e., the process of expansion and contraction of
answer set. Hereby agents revise its own initial demands and update its negotiation program. In
order to make the negotiation success, the buyer must accept the seller’s fact demands $Fact(\Pi_k)$,
if his demands conflict with the seller’s fact demands $Fact(\Pi_k)$, he must give up these demands
and try to accept the seller’s fact demands $Fact(\Pi_k)$, then the buyer can selectively accept the
seller’s other rules. Suppose a sub-negotiation $N(\Pi_k, \Pi)$, $\Pi$ accepts $\Pi_k$’s fact rules is denoted by
$accept(\Pi, \Pi)\rangle$, giving up demands means a set of literals eliminated from an ASP in some
way, this is the process of forgetting, and now we will introduce the notion of strong forgetting
type [11], forgetting a literal $L$ from the buyer’s ASP $\Pi$ is denoted as $SFrget(\Pi, L)$, which is
obtained from the following transformation:

- For any pair of rules $r$ and $r'$ in a answer set program $\Pi$, $Head(r) = L$ and $L \in Pos(r')$, 
  add rule $\{Head(r') \leftarrow Pos(r), \{L\} \} not \text{Neg}(r), \{not \text{Neg}(r')\}$ to $\Pi$;
- Delete all rules $\{r|L \in Pos(r)\}$;
- Delete all rules $\{r|Head(r) = L\}$;
- Delete all rules whose body contain $not L$. 

For a set of literals $S$, forgetting $S$ from an ASP $\Pi$ is denoted as $SFroget(\Pi, S)$.

In the sub-negotiation process both the buyer and the seller set some preferences on literals, these literals which represents their negotiation demands all have different utility to them. The existence of preferences can help them make a better choice on answer set and decide whether accept the opponent’s literals or not. Now we define a utility function based on these literals, and assign a utility to each literal in the answer set that we choose as demands set. Obviously, in a sub-negotiation process, we only need to concern about the buyer’ utility functions, suppose his utility functions is $\mu_B$, and at the start of negotiation, the total utility of the buyer is equal to $a$.

**Definition 2 (Utility function)**

- For an buyer’ ASP $\Pi$, his negotiation preferences $B$ is according to the selected answer set of his ASP $\Pi$, and $ASP(\Pi) = \{b_1, b_2, ..., b_n\}$, each literal represents a subject of a buyer’s preference, and a utility function $\mu_B : B \rightarrow \mathbb{R}^+$ assigning a utility to each of them, such that $\sum_i \mu_B(b_i) = a$.

For two given ASPs $\Pi_k$ and $\Pi$, respectively represents the negotiation knowledge base of the seller and the knowledge base of the buyer in a sub-negotiation. At the start of negotiation, the buyer try to accept $Fact(\Pi_k)$, if $\exists L((\neg L \in Fact(\Pi_k)) \land (L \in AS(\Pi \cup Fact(\Pi_k))))$, then $SFroget(\pi, L)$, obviously, if the buyer’s demands that conflict with the seller’s fact demands, in order to reach an agreement, the buyer must give up these demands and try to accept the seller’s fact demands. Of course, giving up demands leads to the decrease of the total utility and below the initial total utility. The buyer as a rational agent, he will try to accept the seller’s other rules so as to make up for the decrease of the total utility. For new literals that appear in the buyer’s answer set, the buyer assigns to a new value to these literals according to his own utility function.

$\sum_i \mu_B(b_i) = a$. 

Given negotiation is a decision-making procedure in which each agent chooses a deal from the negotiation set [12]. In order to make negotiation terminate successfully, the buyer should select a deal from the negotiation set. The buyer of course will select a deal that brings him maximum utility after the seller put forward all his negotiation rules. For the reason that bargain can be viewed as not a stringent conception of zero-sum game, and a zero-sum game is a mathematical representation of a situation in which a participant’s gain (or loss) of utility is exactly balanced by the losses (or gains) of the utility of the other participant(s) [13], the increase of the buyer’ total utility often is company with the decrease of the seller’s benefit, the seller also as a rational agent, he can put forward every rules that are not facts at regular intervals. Let $\mu_B(t)$ denotes the utility of the consumption of time, if the buyer is patient and the consumption of time is doesn’t matter, i.e. $\mu_B(t) = 0$, he will likely to select a deal that bring him maximum utility. For a buyer who is impatient, the expense of time to find a deal that bring him maximum utility may exceed the benefit of accepting new rules, i.e., $\mu_B(t) + \mu_B(R) \leq 0$, then he will not try to accept more rules.
3.4 Update of the the seller’s ASP

A knowledge base must be updated when new information arrives. There are three cases in updating a knowledge base. The first one is that a knowledge base contains two different kinds of knowledge-variable knowledge and invariable knowledge. The second one is that there is no such distinction and the whole knowledge base is subject to change. As the third one, suppose a knowledge base containing inconsistent information. In this situation, a knowledge base must be updated to restore consistency by removing the source of inconsistency [14].

In our negotiation solution, the update of the seller’s decision-making knowledge base Fact($\Pi_v$) corresponds to the first case. Suppose Fact($\Pi_v$) regarded as core requirement areas of the seller are invariable and other rules are variable. After the above negotiation, the seller can have a set of literals that synthetically consider the vast majority of the buyer’s demands, such that common literals of all buyers, these literals are reference factors when the seller decide to updates his decision-making knowledge base Fact($\Pi_v$). Certainly, if these common literals of all buyers conflict with the seller’s fact, the seller doesn’t update his ASP. Next, we will illustrate the update of the seller’s decision-making knowledge base Fact($\Pi_v$) when introduce a literal of the opponent, the following definition describe the condition that must be fulfilled.

**Definition 3** Let $P$ be the ASP of the seller, and $G$ a literal. Then, a program $P'$ accomplishes an update for the insertion of $G$ to $P$ if

- $P'$ is consistent.
- $P' \models G$.
- Fact($P$) $\subseteq$ Fact($P'$).
- $[P' \cap (P \setminus$ Fact($P$))] $\subset$ $[P'' \cap (P \setminus$ Fact($P$))].$

In the above, the first condition implies that the answer set of the updated ASP doesn’t exist complementary literals, the second says that the accepted literal must be a consequence of the resulting ASP. The third condition presents that the invariable part of $P$ is unchanging, while the fourth condition presents that $P'$ minimally changes the variable part. Definition 3 illustrate the condition in the case of introducing a new literal into the ASP of the seller’s decision knowledge base. As for how to make decision on a literal, we will use dependency semantics [15] to solve it. An ASP can been seen as a restricted version of default logic program, a default rule is applicable if its premise is satisfied and the assumption is valid [16]. In ASP, assumptions are default negated literals and are given by the negative body of rules. In general, the addition of new information can falsify assumptions that consequently blocks the applicability of rules.

For a literal $L$, its dependency relation can be denoted by $d = (L, (M, N))$, where $M$ is the set of premise and $N$ the set of assumption that this literal depends on in this ASP, as how to get the promise and assumption of a literal form an ASP, we can refer to the literature [15]. For the seller’s ASP $\Pi_v$, $\Pi'_v$ accomplishes an update for the insertion of $G$ to $\Pi_v$, if $M \in AS(\Pi'_v)$ and all literals in $N$ are falsified, then the seller can make decision on $L$. Then we will illustrate this process by an example as follows:

```
Caporal ← .
Raise_price ← High_Quality; High_price.
```
\[ \text{Cut\_price} \leftarrow \text{Caporal}; \text{not High\_prise}. \]
\[ \text{Price\_unchanged} \leftarrow \text{Caporal}; \text{not Negative\_comment}. \]

\(\text{AS}(P) = \{\text{Caporal, Cut\_price}\}\), the promise and assumption that literal \(\text{Cut\_price}\) depending on is respectively \(\{\text{Caporal}\}\) and \(\{\text{not High\_prise}\}\). For the reason that the assumption set of \(\text{Cut\_price}\) is not a null set, the seller can’t make decision on the literal \(\text{Cut\_prise}\). Suppose the previous transaction records indicate that this goods are greatly appreciated, and \(\text{High\_prise}\) is added into \(P\), we can get a new ASP \(P'\):

\[ \text{Caporal} \leftarrow . \quad \text{High\_prise} \leftarrow . \]
\[ \text{Raise\_price} \leftarrow \text{High\_Quality, High\_prise}. \]
\[ \text{Cut\_price} \leftarrow \text{Caporal, not High\_prise}. \]
\[ \text{Price\_unchanged} \leftarrow \text{Caporal, not Negative\_comment}. \]

\(\text{AS}(P') = \{\text{Caporal, High\_prise}\}\). For \(\text{Cut\_price} \notin \text{AS}(P')\), we conclude new literal \(\text{High\_prise}\) blocks the applicability of rule \(\text{Cut\_price} \leftarrow \text{Caporal, not High\_prise}\). As for a Caporal goods, the seller chooses either cutting price or keeping price unchanged, since he can’t choose to cut price, he can only choose keep price unchanged.

4 Conclusions and Future Work

In this paper, we have developed a logical framework of asynchronous negotiation and update based on ASP. In the framework, a special negotiation scenario in Electronic Commerce distinguishingly discussed, a negotiation procedure is divided into two stages. The first is negotiation process between the seller and the buyer, the second is the update of the seller’ ASP. In general, the update of the seller’ ASP can help the seller to make decision on goods, we use dependency semantics to analyze this decision-making process. Given the limitation of only one goods is concerned in our work and a buyer may buy a variety of goods in a transaction, we are going to extend this framework to a negotiation of a variety of goods.

Acknowledgement

We should thank the anonymous referees for their comments. This work was supported in part by the National Natural Science Foundation of China (61262029) and the Major Project of National Social Science of China (14ZDB016).

References