Title: **Learning Drifting Negotiations**

Fabrício Enembreck[1], Bráulio Coelho Ávila[1], Edson E. Scalabrin[1], Jean-Paul Barthès[2]

[1]PUCPR, Pontifical Catholic University of Paraná
PPGIA, Graduate Program on Applied Computer Science
R. Imaculada Conceição, 1155
Curitiba PR – Brazil
Phone/Fax: + 55 41 3271-1669
{fabricio,avila,scalabrin}@ppgia.pucpr.br

[2]UTC- Université de Technologie de Compiègne
UMR CNRS HEUDIASYC - Centre de Recherches Royallieu
BP 20529 - 60200 Compiègne, France
Phone/Fax : + 33 (0)3 44 23 44 77
barthes@utc.fr

Abstract

In this work we propose the use of drift detection techniques for learning offer policies in multi-issue bilateral negotiation. Several proposals aiming at the development of *adaptive trading agents* have been made. Such agents are capable of learning their competitors' utility values and functions, thereby obtaining better results in negotiation. However, learning mechanisms they generally used disregard possible changes in a competitor's offer/counter-offer policy. In that case, the agent performance may decrease drastically. The agent then needs to restart the learning process, as the model previously learned is no longer valid. Drift detection techniques can be used to detect changes in the current offer model and quickly update it. In this work we demonstrate with simulated data that drift detection algorithms can be used to build adaptive trading agents and that they offer a number of advantages over other techniques. The results obtained with the IB3 algorithm (Instance-based) show that the agent performance can be rapidly recovered when changes interesting to the competitor are abrupt, moderate or gradual.

1. Introduction

In a negotiation process, two or more players exchange offers and counter-offers concerning goods, resources or services needed for achieving a goal. It follows that some degree of learning skills is fundamental to the improvement in negotiation strategies. For example, one might consider the following scenario: Anna needs to insure her flat. The policy cost is calculated on the following items: (i) fire (10%), (ii) natural disasters (10%) and (iii) burglary (80%). However, she is not aware of this scheme, for it is part of the internal strategy of the insurance company. Despite that, after several interactions with the insurance company, she notices that the capital covered for robbery has a significant impact on the total cost. Based on such findings, she decides to drastically reduce coverage against burglary for her building is fairly secure. Such a decision allows her to save a great deal of money. The following year, Anna asks for the renewal of her policy under the same conditions, however her bank account reveals she was charged twice the amount of the previous year. Outraged, she goes to the insurance company and is informed that there is nothing else she can do, since the policy has already been processed and bank charges have already been programmed. To explain the amount she was charged, the company states that there are a number of weather phenomena (heavy rains and hurricanes) that are predicted for the coming year, which changes the weight of natural disasters from 10% to 60% of the policy cost.

This scenario indicates two important impacts of learning on negotiation: (i) learning allows discovering negotiation strategies that can bring important benefits (Anna saved a considerable amount of money when she paid for her first insurance policy); and (ii) the concepts acquired in a negotiation may become obsolete and have undesirable effects (Anna spent a great deal of money in the renewal of her insurance policy), thereby raising

the need for a mechanism to detect and automatically update such concepts. Such an issue can be tackled with adaptive intelligent agents.

Researches on models and applications of negotiation using (human or computer) agents are quite frequent. According to Faratin et al. (1998), negotiation is a process whereby two or more agents are required to come to a decision in order to resolve a conflict. When both agents are involved in transactions to attain a common goal one talks of *Cooperative Negotiation*. A negotiation mechanism is composed of two main elements: the *negotiation protocol* and the *negotiation strategy*. The negotiation protocol defines the meaning of each interaction among agents, what can be done and which sequences of actions may be taken (Fátima et al, 2004). The negotiation strategy is usually composed of a set of plans that must be pursued by an agent so it can achieve its goals. Negotiation processes can be used in a variety of domains involoving distributed problem solving. Within an environment with limited resources like e-commerce, negotiations between customer and supplier agents can lead to coordinated forms of allocating and saving resources, meeting values that mutually satisfy each agent in the supply chain. Whenever there are multiple criteria for evaluating offers and counter-offers in a negotiation process (which is the case in the previous scenario), the process is said to be multi-issue.

A number of projects have been carried aiming at developing intelligent trading agents for learning their competitors' preferences, functions or utility values. Usually, this information is private to each agent involved in the negotiation process and its discovery increases the probability of optimizing resources and obtaining advantages. Thus, probabilistic algorithms (Zeng and Sycara, 1998) (Buffet and Spencer, 2005), reinforcement learning (Huang and Lin, 2005), genetic algorithms (Lau, 2005) or kernel density estimation (Coehoorn and Jennings, 2004) have been studied. Nonetheless, most proposed algorithms

do not satisfy some of the constraints inherent to the problem of discovering offer policies in negotiation: (i) the learning process must be efficient, allowing to propose values satisfying the trading agents after a few interactions only; (ii) it must also be computationally efficient, as the agent has limited processing and response time capabilities; (iii) the learning algorithm must be capable of learning complex relationships among offers once it faces a multi-issue negotiation; therefore, algorithms that perform local searches may miss important relationships among attributes of offers; (iv) the learning process must be incremental, because besides the limited number of observations, an agent function and utility values may be modified during the negotiation process, and there can be interferences from external factors as well. In this work we concentrate particularly on the latter problem, proposing the use of *drift detection techniques* for online, incremental learning of a trading agent offer policies. This allows an agent to automatically detect changes in its opponent's negotiation strategies. We start from the idea that the negotiation process is bilateral and multi-issue.

Drift detection (Helmbold and Long, 1994) (Gama et al, 2004) is a widely researched problem in the machine learning community. Through simulations, we demonstrate in this work that drift detection techniques are potentially useful for discovery of offer policies in negotiation.

The drift detection techniques studied in this work are derived from instance-based learning. It can be verified in (Aha et al, 1991) that someof the proposed algorithms make rather efficient use of processing time and memory space, and can be easily implemented. We chose the IB3 algorithm proposed by Aha, because according to the author it shows reasonable robustness against noise and irrelevant attributes. Such characteristics are

important for negotiation, as agents can distinguish real changes from noise besides the possibility of exploring relationships amongst the various offer issues. Updating is also a low cost process in the algorithm, which allows making rapid decisions and identifying changes in an agent offer policies. Therefore, our goal consists in proposing a technique that allows an agent to identify changes in negotiation policy of a competitor agent and to classify an arbitrary offer as either interesting or not.

In the following section we discuss some ideas related to offer policy learning, as well as the most important issues related to this problem. We characterize the proposed problem as a drift detection approach by discussing its most relevant theoretic aspects in Section 3. We present our technique in Section 4 and discuss experimental results in Section 5. Our conclusions are given in Section 6.

2. Offer Policy Learning and Related Work

With the growth of Internet and the democratization of technology, it can be observed that the number of service and product suppliers, and, specially, potential customers grows exponentially every year. Nevertheless, technology must favor the establishment of relationships amongst such players in a way that both customers and suppliers achieve their goals, like minimizing costs, acquiring goods safely, guaranteeing a large number of customers, or resolving problems of demand and stock. One way of favoring the growth of such alternatives for business employs trading agents. Trading agents can play the role of customers or suppliers depending on the context. The automatic discovery of offer policies in negotiations is a fundamental issue for the evolution of e-commerce technologies, as trading agents must participate in negotiations and business transactions in order to maximize resources in favor of some customer or supplier. A trading agent must implement

at least two main components: an *interaction protocol* and a *negotiation model*. The interaction protocol defines the meaning of each interaction between agents, what can be done and which sequences of actions should be employed (Fátima et al., 2004). The negotiation model comprises the description of the offers and issues, determining value domains for issues and, more importantly, a reasoning model, usually based on a utility model. The reasoning model allows the agent to reason about offers and counter-offers (Faratin et al., 1998), thereby defining specific priorities and behaviors. In recent years some researchers have proposed intelligent trading agents provided with an extra module composed of a learning model allowing the agent to learn their competitors' utility function. That function helps the agent to define its own goals so as to produce offers that satisfy its needs more quickly. For instance, one could considered, a buyer agent that needs to receive a large quantity of an arbitrary product immediately. If few suppliers are known to be capable of guaranteeing the volume to be delivered, the vendor may guarantee immediate delivery at a high price, provided that it is not higher than the competitors' price. Of course it depends on knowing the customers' limitations, which generally is confidential information. That information can be discovered by observing similar behaviors in previous negotiations using pattern recognition techniques and machine learning. Several projects with that goal have been proposed. Genetic algorithms have been used by Gerding and Bragt (2003) and Lau (2005) in attempts to find the best offer and counter-offer policy. However, such techniques are hardly ever applicable to e-business due to the complexity of the involved calculations. Probabilistic techniques based on Bayes model have also been used with relative success in (Zeng and Sycara, 1998) or (Buffet and Spencer, 2005). The proposed solutions however, suffer from problems related to the distribution of data, which is rarely known, and the volume of data, which is usually insufficient for the discovery of

effective probability functions. Moreover, a pre-existing dataset is rarely available to the estimation of probabilities *a priori*. Orthogonal polynomials have been used for estimating the utility functions in trading agents by (Saha et al., 2005). Nonetheless, even for limited negotiation spaces, this technique needs a large number of interactions to discover efficient utility functions. Coehoorn and Jennings (2004) have proposed a technique based on kernel density estimation to discover the weights of each competitor's issue. Based on such weights, the authors used the algorithm proposed in (Fatarin et al., 2002) to estimate trade-offs that satisfy the agents involved in the negotiation.

Although some of the cited projects gave satisfactory results, they do not consider an important aspect of negotiation: there are various cases where the negotiation process cannot be modeled as a stationary function, but as one under the influence of factors that may or may not be related to the negotiation itself. This means that agent utility values and functions may change in time, for example due to its poor performance or because of the occurrence of unexpected events, alien to the negotiation environment. As an example, a sudden lack of raw material for a supplier will seriously affect the way its sales agent negotiates with customers. Likewise, a customer who notices his income is decreasing in time may gradually alter his offer policies in order to reduce the need for immediate capital. Such characteristics make most learning methods that we mentioned inappropriate since they are unable to adapt the learned model to changes dynamically. The agents must, therefore, restart the learning process after a long period during which their performance is poor and does not reflect the brand new competitor's negotiation model. The identification of changes in concept descriptions is a task known as *drift detection* (Schlimmer and Granger, 1986) (Helmbold and Long, 1994) in Machine Learning (Mitchell, 1997). In this

work, we focused on detection of changes in negotiation policies by describing how *drift detection* algorithms can contribute to the solution of this problem. Next section discusses the main concepts related to this subject.

3. Drift Detection

In this section, we present the drift detection problem and discuss some algorithms used for solving it.

3.1 Drift Detection Definition

In domains where the environment constantly changes or data is continually produced, dynamic learning techniques need to be used as soon as the target-concepts of the algorithm change as time elapses. Such modifications may be abrupt or occur slowly. Nevertheless, most machine learning and data mining algorithms assume that training sets are generated according to a stationary distribution of a probability function. Those algorithms are, therefore, incapable of handling the problem we want to solve. Gama et al. (2004) mention other problems that involve detection of changes in target-concepts like biomedical and industrial processes monitoring, fault detection and diagnostic, or complex security systems.

Detecting changes while learning dynamic class models is the problem known as drift detection (Kilter and Maloof, 2005) (Stanley, 2003). The following paragraphs characterize the concept of drift detection as described by Stanley (2003).

For simplicity, we defined as a *concept* a normal disjunctive formula that relates a set of qualitative variables like Price $\in$ {very_low, low, normal, high, very_high}. Thus, a *concept* may be described as, for instance, "Price = (very low or low) and Quality = high." With such a description one can easily verify whether an instance is covered by an arbitrary

concept by checking if the conditions in the corresponding concept description are satisfied by that instance. Drift detection may be defined as a modification in the current concept. In order to illustrate such modification we need a couple of concepts. Let *A* and *B* be two concepts and i_1 to i_n a set of instances. Until reaching instance i_x, the current concept *A* is stable. After a certain number Δx past i_x, the concept *B* takes the role of current concept. The change from concept *A* to concept *B* is smoothly taking place between instances i_x and $i_{x+\Delta x}$ as shown Figure 1.

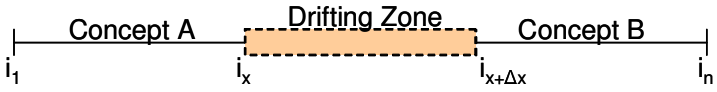

Figure 1. Drifting from A to B.

When $\Delta x = 1$, an abrupt change from *A* to *B* is said to have occurred. On the other hand, when $\Delta x > 1$, there exists a drifting zone and the modification proceeds gradually. The drifting zone can be modeled as a probability function α which defines the dominance of concept *A* over *B*. Thus, $p(A) = \alpha$ and $p(B) = 1 - \alpha$. In order to measure the probability of concept *A* in an arbitrary $i_c \in \{i_x , i_{x+1} , ... , i_{x+\Delta x}\}$ one needs to use $\alpha = (c - x) / \Delta x$. The equation establishes that the probability of occurrence of a concept *A* decreases linearly as the probability of *B* increases.

3.2 Considerations for Concept Drift

A concept drift technique must include three main goals: (i) rapid detection of concept drift, (ii) distinguishing concept drift from noise; and (iii) recognizing and dealing with recurring contexts. Rapidly detecting a concept change in data is essential. Such a feature enables actions to be selected in a way that minimizes the negative impact caused in the system by mistaken actions. Put simply, that means lesser instances will be incorrectly classified by the system, thereby lesser mistakes will be done. On the other hand, some

systems may receive noisy perceptions due to faults in the input or monitoring devices, or due to partial representations of the environment. Such changes consist of anomalies that must be identified without, however, modifying concepts acquired by the system during the learning process. The work done by (TSymbal, 2004) or (Stanley, 2003) mentions such need. Another important aspect of concept drift techniques lies in the ability to identify recurring concepts, i.e., concepts that alternate between valid and invalid states with some degree of regularity. There are several domains where concepts are naturally recurring, for instance, the cycle of seasons through a year (spring, summer, fall and winter). Recurring concepts directly affect current prediction parameters. They can be stored and reused in future situations. By acting in such a way, the system does not need to repeat the learning process for a previously learned concept; it only needs to identify the valid recurring concept, saving resources, reducing response time and possibly increasing its utility.

3.3 Updating Techniques

In several application domains, data is continuously generated and stored. Moreover, a system for administration of critical situations continuously acts according to input data. Consider for example a network of atmospheric radars that monitor an arbitrary region in order to predict destructive electrical storms. Data transmission frequencies are about hundredths of seconds. Within such an environment, the number of concepts (atmospheric conditions) is limited, while time and confidence in decision making are critical. Therefore, an issue must be considered: how to guarantee that the prediction model of the system will always be up-to-date? To let a system have its concepts continuously providing a consistent representation of data, the use of techniques for online (incremental) learning is recommended. In this case, concepts are only modified; no concept is reconstructed. Modifications of concepts should permit a new observation to be correctly classified. On

the other hand, a procedure for off-line (batch) learning would not be suitable, since it would require the system to store a great number of instances before starting a new execution of the learning algorithm. During the learning process (which may be slow), instances may be incorrectly classified, as the system operates with an obsolete model.

Some systems also need user interaction to support the update process. For instance, when a tool for detection of unsolicited electronic messages (SPAM) intercepts a suspicious message, the system asks the user to indicate whether or not that message constitutes SPAM. Without explicit user feedback, the system is unable to update its knowledge base, which might result in incorrect classification of messages, leting unsolicited messages come through.

3.4 Detecting Concept Drift

Numerous methods have been proposed to deal with the drift detection problem in machine learning. They can be categorized according to their training features as either online or batch. An online learning method modifies its models whenever a new example is observed. On the other hand, a batch method processes a whole dataset once, updating its data models periodically. Online learning is particularly interesting to this work, since we need to consider that offers/counter-offers are received and used in the negotiation. The most ubiquitously widespread drift detection techniques are (i) windowing, (ii) example weighing, according to their age or utility, and (iii) example selection. According to Gama et al. (2004), techniques that deal with drift detection can still be split into two other categories: (i) techniques that adapt the classifier at regular intervals without verifying whether any changes have occurred; or (ii) techniques that detect changes beforehand and then adapt their corresponding classifiers to them. Examples of the former are windowing and example weighing. The hypothesis whereby the importance of an example decreases

with time is the foundation of the example weighing technique. It can also be used as a tool to select the most relevant examples. More sophisticated techniques overcome some the mentioned problems by generating and maintaining a set of classifiers (Kolter, 2005) (Stanley, 2003). Some of them are described in the following sections.

Windowing

Whenever a time window is used, the classifier is reconstructed from the examples within such a window. The complexity of this technique lies on identifying the proper size of the temporal window: excessively small ones ensure rapid adaptation, but cannot identify relationships among variables due to the partitioned view of data, whereas extremely big ones produce good classifiers for stable zones to the detriment of losing the capacity for identifying changes quickly. If some change is detected at any instant of the monitoring phase, some actions may be triggered in order to let the classifier adapt accordingly. For instance, modifying the window size in order to accommodate the set of examples affected by that change will do that. In addition, the window size may be reduced upon the detection of a change and enlarged while no change is observed.

The windowing technique obeys the following principle: a storage area (or buffer), called *Window*, is created to serve as a container for the N most recent observed instances. The learning algorithm uses the N instances in the window to learn the current data model. The window size may be fixed or variable. An advantage for the variable one is that it may adjust the number of instances that will be used in the pattern discovery process as a function of the stability of the data distribution. For instance, when the distribution of data undergoes a stable period, i.e., when no concept change has been observed recently, the bigger the window, the better will be the model generated by the algorithm to represent it. Conversely, when the distribution undergoes a transitional period, the smaller the window,

the better will be the system performance, as instances display significant variations to one another [Klinkenberg and Renz, 1998].

In [Stanley, 2003] a comparative study on windowing is presented along with other strategies for detection of concept drift. In the study, the authors experienced difficulties in their attempt to automatically adjust the window size. All strategies imply a slow change in window size, which requires the algorithm to experience over relatively long periods with a high error rate in classifications [Klinkenberg and Renz, 1998].

Error Estimation

The strategy for error estimation consists in using the error rate of the algorithm on the instances received by the system to identify concept changes. As shown Figure 2, this strategy employs two thresholds, namely Warning Level and Drift Level. Such thresholds are used by the concept drift detection algorithm to indicate that the error rate for classification lies within an interval that denotes a possible concept change. Thus, when the error rate overtakes the Warning Level, additional instances are stored in a buffer. When the error rate reaches the Drift Level, current concepts need to be updated and the learning algorithm starts. All the instances stored from the instant when the Warning Level was overtaken until the one when the Drift Level was reached become the training dataset for the algorithm [Maloof, 2002].

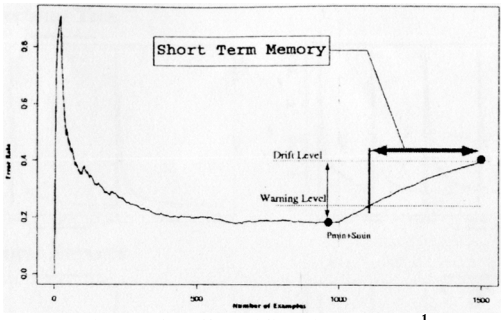

Figure 2 - Classification Error[1]

Figure 3 shows the variation in error rate through time in a test database where several different concepts define the distribution of data. Upon each concept change, a detection occurs and the system immediately updates its model. Each peak on the chart represents a concept change, though it can be easily noticed that the error rate is rapidly reduced, indicating that the strategy responds adequately to the concept change.

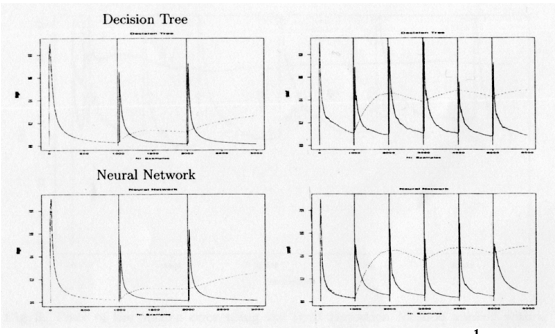

Figure 3 – Error rate through time[1]

Concept Drift Comittee (CDC)

The CDC [Stanley, 2003] is an alternative for algorithms that use time windows. These algorithms require heuristics to decide when window sizes should be altered, whereas CDC does not. The CDC algorithm relies on a poll to detect the Concept Drift and update concepts.

Consider a committee C, comprising n hypotheses, where each hypothesis is a decision-tree built upon all instances it knows (s_i). An incoming instance is evaluated by

[1] Extracted from [Gama et al., 2004]

each hypothesis; the ones that incorrectly classify the instance have their weights reduced until they are removed from the committee and replaced by new hypotheses. Initially, C is empty (no hypotheses). Whenever a new instance arrives, if the number of hypotheses in the committee is less than a maximum n, a new member is added to it. The newly created member h_1 initializes its set s_1 with the current instance only. Thus, upon the arrival of the second instance, the committee C contains the hypothesis h_1 only, whose training set is s_1, which holds the instance i_1.

At the time the instance i_2 arrives, a new member h_2 is added to the committee and its training set s_2 is composed of the instance i_2 only. At this moment, the committee has two hypotheses:

- h_1, with s_1 comprised of $\{i_1, i_2\}$
- h_2, with s_2 consisting of the instance i_2 $\{i_2\}$

Members are added to the committee until the maximum number of members (n) is reached. For the committee members (hypotheses) to be evaluated, each of them must vote on test instances that are derived from the current distribution (target concept). The weight of each vote is equal to the performance of the corresponding member over the last m training instances. For this reason, members whose recent performances have been satisfactory participate with votes of higher weight. When a member's performance falls beneath a threshold t, it is replaced by a brand new one that has no knowledge of past instances. Therefore, the committee will always be representing the current distribution of data.

In periods of stability, the oldest members are more reliable, though when there are transitions between concepts, members are constantly being replaced. When a new member is added to the committee, it does not contain enough instances to gain considerable weight

for its vote; therefore, it must achieve a certain degree of maturity before it can vote and be replaced (if necessary) by a new member. Such a mature age is reached when the member has already observed *m* instances throughout its existence. Before that, its vote weight is 0 (zero). The work of Stanley [2003] describes the CDC algorithm in details.

It is possible to consider that CDC uses pseudo time windows, as each member is trained with an instance set different from the ones that the other members use. This characteristic allows CDC to exhibit good performances in both gradual and abrupt *Concept Drift*. It should be emphasized that, in the latter case, usually almost all members are replaced, while in the former case they are replaced gradually. Such behaviors make the algorithm noise tolerant and efficient for the detection of *Concept Drift*.

Instance-Based Concept Drift Detection

In this work we use a well-known drift detection algorithm: the IB3 algorithm, proposed by Aha (1991). As stated by the author, the algorithm presents a relative robustness against noise and irrelevant attributes. Such characteristics are quite valuable in negotiation, as agents can distinguish real changes from noise besides the possibility of exploring relationships amongst the various offer issues. Updating is a low cost process in that algorithm, which enables responsiveness for decisions and thus detection of changes in an agent's offer policies. IB3 is a member of the instance-based algorithms family. IB1 corresponds to the standard nearest neighbor algorithm. Although instance-based learning algorithms are capable of reproducing complex discriminating functions, they usually require massive memory resources and processing time for the classification process due to the need to compare and contrast the instance to be classified with all available instances. IB2 attempts to reduce the problem by introducing a criterion for the storage of a new

instance. When a new instance is classified, its storage in the concept description is only performed if its classification was incorrect, otherwise it is discarded. Although saving memory and processing resources, experiments have shown that IB2 is fairly sensitive to noise in data. IB3 has evolved from IB2. Experiments demonstrated that the method is quite robust when applied to data with such characteristics.

IB3 maintains a classification record with each and every instance stored. This record quantifies the classification rate of an instance, i.e., the accuracy rate of the instance in cases where it has been used to classify a new one. That information is used to estimate its future precision. The algorithm also uses a significance test to determine whether an instance either should be relevant for future classifications or constitutes noise. In the latter case, it is discarded. The algorithm stores an instance if its precision is significantly higher than the observed frequency of its class, and removes it if its precision is significantly lower. A detailed description of the algorithm can be found in (Aha, 1991).

4. Drift Detection for Drifting Negotiation

In this work we propose using drift detection to discover negotiation policies without the need of the best available algorithm. We believe that such a technique constitutes an innovative solution to the proposed problem. We are not, however, interested in proposing and/or using complex methods of negotiation, since our objective is to propose a generic technique that can be used with any negotiation model. In addition, the use of a specific utility model might introduce biased results, rendering it invalid for other negotiation models. We propose a generic technique allowing an agent to identify changes in a competitor's negotiation policy, classifying an offer as either possibly interesting or non-interesting to its competitor. This partial information may be used along with a strategy for

exploring the space of possible offers in order to discover an interesting offer and to lead the agent to a better negotiation. Figure 4 shows the integration of the proposed drift detection model. Upon receiving an offer/counter-offer, the agent classifies it as interesting in case it has been accepted by the corresponding peer and non-interesting otherwise. Based on that offer, a classified instance is then constructed and transferred to the drift detection module, where it can be stored in the concept description database or rejected. Next, the negotiation module generates an offer/counter-offer that is iteratively classified in the drift detection module. As soon as an offer/counter-offer is classified as interesting, the iterative process ends and the offer is delivered to the peer agent.

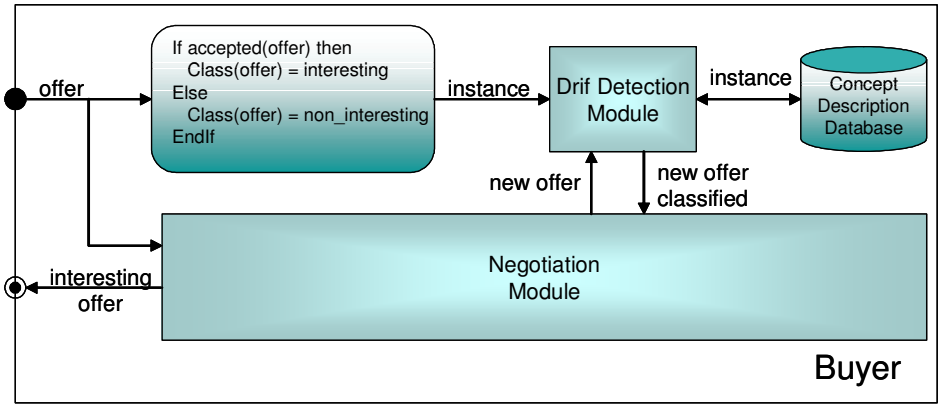

Figure 4. Architecture of an Adaptive Trading Agent.

4.1 Defining the Offer Space

To illustrate the technique we need to define the offer space. For the sake of simplicity, we defined issues as qualitative variables. They may or may not be ordered. Table 1 presents each of the issues used in this work.

Table 1. Description of Issues.

Issue	Domain
Color	[black, blue, cyan, brown, red, green, yellow, magenta]
Price	[very_low, low, normal, high, very_high, quite_high, enormous, non_salable]
Payment	[0, 30, 60, 90, 120, 150, 180, 210, 240]

Amount	[very_low, low, normal, high, very_high, quite_high, enormous, non_ensured]
Delivery_Delay	[very_low, low, normal, high, very_high]

4.2 Defining Concepts

To contextualize the definition of concepts we employed a bilateral negotiation between a buyer and a sales agent. The former will always attempt to predict whether an arbitrary instance constructed upon an offer will be either interesting or non-interesting to the latter. We have defined some hypothetical, disjunct concepts that a vendor can use and whose definitions the buyer must discover. Table 2 illustrates such a sequence of concepts, that represents an abrupt drift.

Table 2. Description of 5 concepts.

Concept ID	Description of an interesting offer	Concept Description
1	(Price = normal **and** Amount = large) **or** (Color = brown **and** Price = very_low **and** Delivery_Delay = long)	An offer is judged interesting in case its price is normal and the amount of the desired product is large. However, a problem in the production line that handles brown products requires the vendor to reduce price for delivery delays will probably be too long.
2	(Price = high **and** Amount = very_large **and** Delivery_Delay = very_short)	Suppliers face difficulties in providing raw materials to the company and production has thereby fallen drastically. This lack of product requires the company to penalize customers who buy large quantities and need them immediately, with a rise in prices for delivery under such conditions.
3	(Price = very_low **and** Payment = 0 and Amount = large) **or** (Color = red **and** Price = low **and** Payment = 30)	Important customers have declared incapacity to honor big contracts and the company needs capital to continue operations. Therefore the company will sharply reduce prices of all products for large quantity orders and cash payment. Nevertheless, product red is marginally on demand, which prevents its price from falling too much. To compensate that, the company decides to extend its payment delay to 30 days.
4	(Color = black **and** Payment = 90 **and** Delivery_Delay = very_short) **or** (Color = magenta **and** Price = high **and** Delivery_Delay = very_short)	The company maintains large amounts of product black, implying high costs of storage and logistics. The company then extends the payment delay for that product, guaranteeing immediate delivery. Product magenta is now made of imported raw material, having its cost increased.

		Nonetheless, there is also a large amount of this product in stock that needs to be sold quickly.
5	(Color = blue **and** Payment = 60 **and** Amount = little **and** Delivery_Delay = normal) **or** (Color = cyan **and** Amount = little **and** Delivery_Delay = normal)	Big investors are now endowing the company, increasing its capital resources. It then starts to aggressively position itself to gain market share by popularizing some of its products. The company guarantees long payment delays for products blue and cyan, even for small quantity orders.

4.3 Simulating Negotiation

Considering the definitions of concepts in the previous section, we used a procedure for generating instances similar to the Stagger system (Wider and Kubat, 1996) and used in (Stanley, 2003) and (Gama et al., 2004). For each concept, 100 instances are generated and used to assess the precision of the system. Next, an arbitrary number of training instances x is generated. The drift detection model then receives one training instance at a time and is assessed on the test set. We determined that a training or test instance can have its class labeled as interesting or non-interesting. This procedure is illustrated Figure 5.

```
Foreach concept c ∈ [1,...,C] do
        generate test instances according to concept c
         Foreach instance i ∈ [1,...,x] do
                learn i with the drift detection approach
                evaluate the performance over test instances
        endForeach
endForeach
```

Figure 5. An abrupt drift detection evaluation algorithm.

Concept drifts can also occur moderately. In this case, each successor concept maintains some degree of similarity to the previous one. To illustrate this situation we defined a total of 8 concepts that are described in Table 3. For the definition of such concepts we used the attributes *Delivery_Delay* that is constant, and *Amount* only. This sequence of concepts can illustrate, for example, the constant increase in the vendor's capacity of production.

Consequently, the vendor is capable of increasing the amount of products delivered with short delay.

Table 3. Moderate Drift Concepts.

Concept ID	Description of an interesting offer
1	(Delivery_Delay = very_low **and** Amount = very_little)
2	(Delivery_Delay = very_low **and** Amount = little)
3	(Delivery_Delay = very_low **and** Amount = normal)
4	(Delivery_Delay = very_low **and** Amount = large)
5	(Delivery_Delay = very_low **and** Amount = quite_large)
6	(Delivery_Delay = very_low **and** Amount = very_large)
7	(Delivery_Delay = very_low **and** Amount = enormous)
8	(Delivery_Delay = very_low **and** Amount = non_ensured)

We also used two concepts *A* and *B* to simulate a gradual concept change in the sales agent. Both concepts are displayed in Table 4. The procedure for gradual drifting generation was described in Section 3.1.

Table 4. Concepts used for gradual drifting.

Concept ID	Description of an interesting offer
A	(Color = blue **and** Delivery_Delay = very_short)
B	(Color = black **and** Price = high) **or** (Color = magenta **and** Payment = 0)

5. Experiments

The experiments we performed show the results of the IB3 algorithm applied to the concepts described in the previous section. The average results obtained with 20 iterations are discussed in this section. Initially, IB3 was evaluated on the existence of abrupt drifts. We fixed the number of instances x = 50 to use with each concept. Figure 6 illustrates the obtained results.

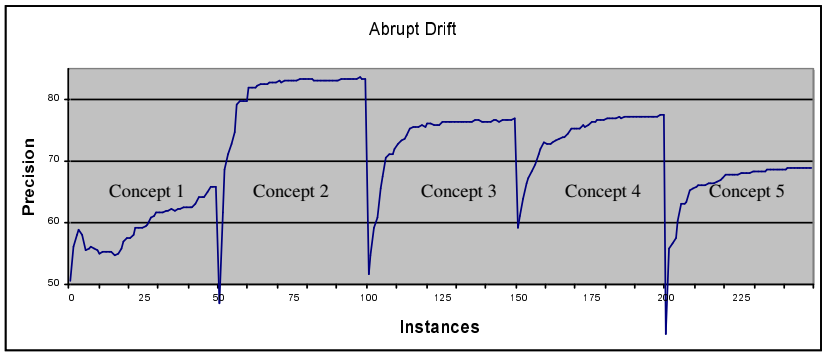

Figure 6. Results of abrupt drift concepts.

It is also noticeable that the detection of drifts occurs quite quickly, usually in less the 10 iterations for each new concept. Another remarkable feature is the performance of the first concept, that was just moderate, probably for more iterations it would have been necessary so that the IB3 algorithm had constructed an adequate representation for concept 1. Such a behavior tends to be exhibited by IB3 due to its conservative model, adopted to prevent the algorithm from learning inaccurate relationships from noisy data.

Results obtained from data with moderate drift shown Figure 7 are quite encouraging. One can clearly notice that falls in performance caused by changes are much less than those identified Figure 6 (less than 10% on average). This behavior results from similarities among the produced concepts. One can also notice an increase in precision for each concept in comparison with its previous peer. However, the increase in precision occurs more slowly than it did when changes were abrupt. As concept descriptions learned by the algorithm present a reasonable performance the algorithm reacts more carefully in order not to remove instances whose absence might decrease the precision of the current model, once the stored instances present good classification accuracy. On the other hand, when the performance is deficient (in case of abrupt changes), performance in the classification of

stored concepts tends to be rather poor and the algorithm executes frequent modifications in the set of instances that form the concept description.

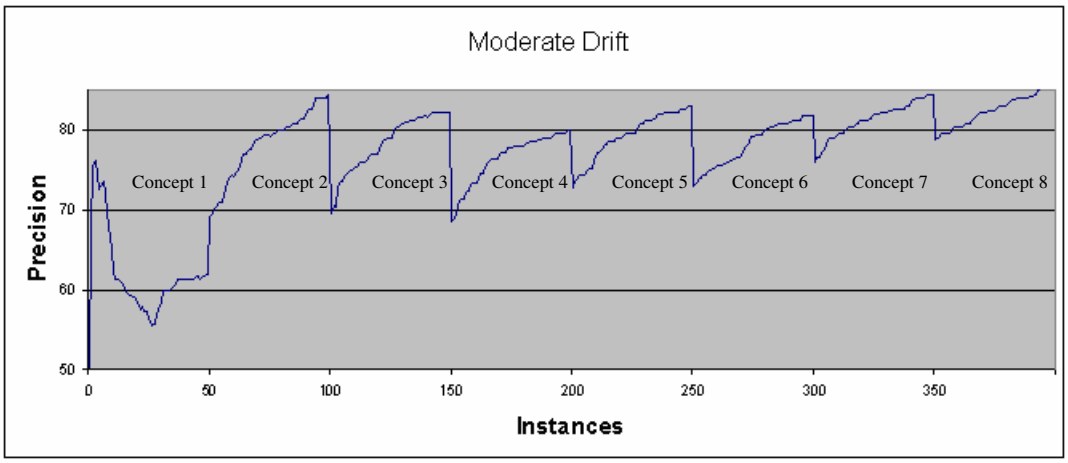

Figure 7. Results of Moderate Drift.

We conducted experiments to assess the impact of gradual drift between a couple of concepts *A* and *B* in a number of instants during the learning process of the buyer agent. Both *A* and *B* were randomly generated. The first experiment aimed at measuring the influence of drift at the beginning of the agent learning process. Figure 8 shows the results obtained with a drift zone ranging from 50 to 150. In this case, the agent performance is minimally affected by changes, maintaining an ascending learning bias. Figure 9 shows that, when the agent has an already consolidated concept description, a short drift (between 150 and 200) only provokes a moderate oscillation. Figures 10 and 11 show that long drift periods tend to attenuate performance oscillations, while maintaining an average performance comparable to the one observed before the start of the drift period (i_x). After the drift period, the tendency for increase in performance can still be amplified, provided that a large number of instances remain available.

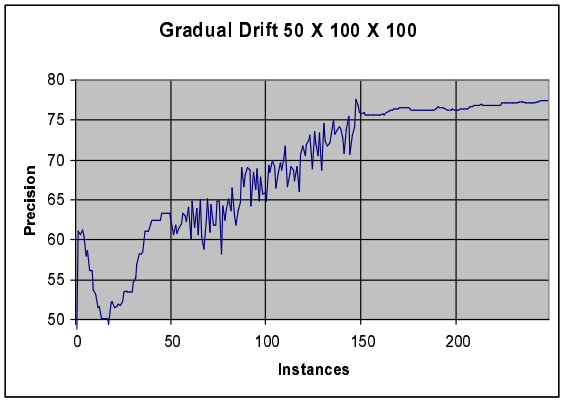

Figure 8. Gradual Drift with drifting zone ∈ [50,…,150].

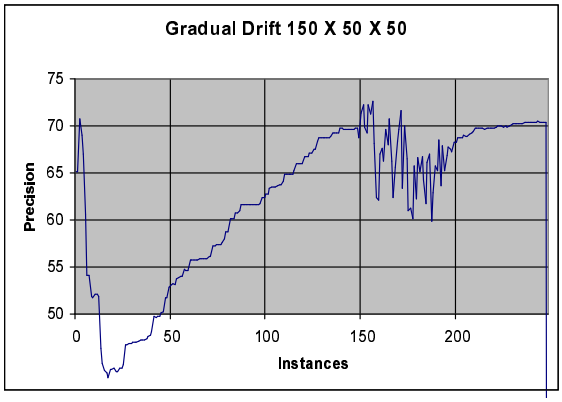

Figure 9. Gradual Drift with drifting zone ∈ [150,…,200].

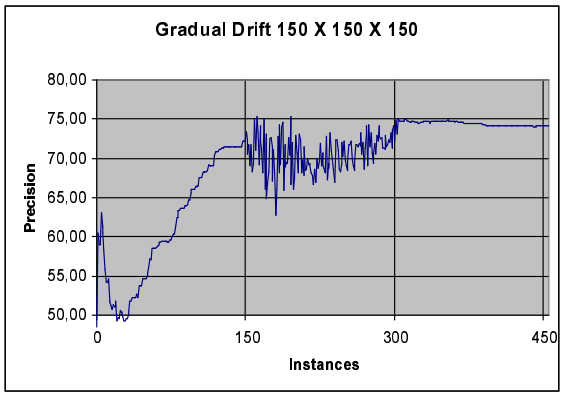

Figure 10. Gradual Drift with drifting zone ∈ [150,…,300].

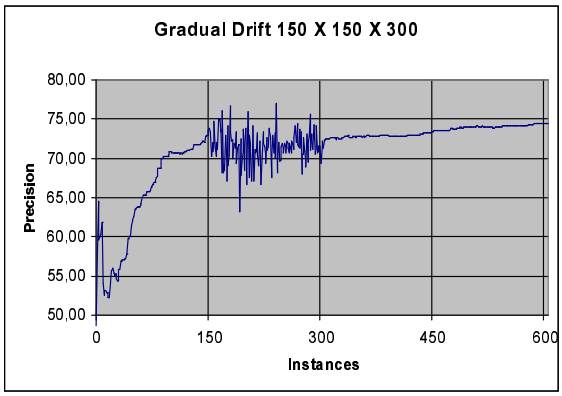

Figure 11. Gradual Drift with drifting zone ∈ [150,…,300] with many instances.

We can state that considering the experiments that have been conducted, algorithms with drift detection capacities could be successfully used to learn offer policies, since results of all experiments show that the learning curve may suffer interferences, though it always maintains an ascending bias. In some cases, detection of changes is quite fast and the agent performance increases quickly.

6. Conclusions

In this work, we demonstrated that algorithms developed for concept drift detection can be successfully used in bilateral and multi-issue negotiations. Recall that our goal was not to propose an integrated model of negotiation (see Section 4), but to show that drift detection techniques satisfy important constraints inherent to the problem of learning offer policies,

such as effectiveness, efficiency, complexity and, mainly, changes in the agent behaviors. Experiments performed on simulated concepts of negotiation showed that the IB3 algorithm can detect changes rapidly and update an agent concept description in order to improve its performance even with a few iterations only. We started from the idea that a negotiation model could exploit the drift model to generate offers that are interesting for both agents involved in the negotiation process, and, as a result, the agent performance in the negotiation process should be proportional to the one obtained by the drift detection model.

In future work, we intend to employ techniques such as the ones described in this work to improve existing negotiation models, similarly to what has been proposed by Coehoorn and Jennings (2005). Once a complete negotiation model is built, we will be capable of assessing the utility values obtained by agents that use different drift detection algorithms. An interesting alternative consists in employing techniques based on ensembles and committees, such as those described by Stanley (2003) and Kolter and Maloof (2005). In spite of a higher computation cost, such techniques can produce fairly satisfactory results for the detection of different sorts of concept drifts.

References

Aha, D.W., Kibler, D., Albert, M.K. 1991. Instance-based Learning Algorithms, Machine Learning. 6(1), pp. 37-66.

Buffet, S., Spencer, B. 2005. Learning Opponents' Preference in Multi-Object Automated Negotiation, Proc. of International Conference on Electronic Commerce'05, pp. 300-305, Xi'an, China.

Coehoorn, R. M., Jennings, N. R. 2004. Learning an Opponent's Preferences to Make Effective Multi-Issue Negotiation Trade-Offs, Proc. of the Sixth International Conference on Electronic Commerce, pp. 59-68.

Faratin, T., Sierra, C., Jennings, N. R. 1998. Negotiation decision functions for autonomous agents, Robotics and Autonomous Systems, Vol 24, N 3-4, pp 159-182.

Faratin, T., Sierra, C., Jennings, N. R., 2002. Using similarity criteria to make issue tradeoffs in automated negotiations, Artificial Intelligence, n. 142 vol, 2, pp. 205-237.

Fatima, S., Wooldridge, M. and Jennings, N. 2004. An agenda-based framework for multi-issue negotiation. Artifcial Intelligence, 152:1, n. 45.

Gama, J., Medas, P., Castillo, G., Rodrigues, P. 2004. Learning with Drift Detection, Proc. of the 17th Brazilian Symposium on Artificial Intelligence – SBIA'04, LNAI 3171, Ana L.C. Bazzan and Sofiane Labidi (eds.), Springer, pp.286-295, Brazil. ISBN 3-540-23237-0

Gerding E., Bragt, D. van, 2003. Multi-issue negotiation processes by evolutionary simulation, validation and social extensions. Computation Economics, n. 22, vol. 1, pp. 39-63.

Helmbold, D. P., Long, P. M. 1994. Tracking drifting concepts by minimizing disagreements. Machine Learning, 14(1):27-46.

Huang, S., Lin, F., 2005. Designing Intelligent Sales-agent for Online Selling, Proc. of the International Conference on Eletronic Commerce, pp.279-286, Xi'an, China.

Klinkenberg, R, Renz, I. Adaptive Information Filtering: Learning in the Presence of Concept Drifts. AAAI-98/ICML-98. 1998.

Kolter, J. Z., Maloof, M., A. 2005. Using additive expert Ensembles to Cope with Concept Drift, 22nd. International. Conference on Machine Learning, Germany.

Lau, R. Y. K. 2005. Adaptive Negotiation Agents for E-business, International Conference on Eletronic Commerce, pp. 271-278, Xi'an, China.

Maloof, M. Incremental Learning with Partial Instance Memory. XIII International Symposium on Methodologies for Intelligent Systems. Lyon, France. 2002.

Mitchell, T. 1997. Machine learning. New York: McGraw Hill.

Saha, S., Biswas, A., Sen, S. 2005. Modeling Opponent Decision in Repeated One-shot Negotiations, Int. Conference on Autonomous Agents and Multi-Agent Systems – AAMAS'05, pp. 397-403.

Schlimmer, J., Granger, R. 1986. Beyond incremental processing: Tracking concept drift. Proceedings of the 5th. AAAI Conference, pp. 502-507, AAAI Press.

Stanley, K. O. 2003. Learning Concept Drift with a Committee of Decision Trees, University of Texas, Department of Computer Sciences Technical Report AI-03-302, September.

Tsymbal, A. The problem of concept drift: definitions and related work. 2004.

Widmer, G., Kubat, M. 1996. Learning in the presence of concept drift and hidden contexts. Machine Learning, n. 23, vol 1, pp. 69-101.

Zeng, D., Sycara, K. 1998. Bayesian learning in negotiation, International Journal on Human-Computer Studies, 48:125-141.