Algorithmic Trading Using Intelligent Agents

Rui Pedro Barbosa and Orlando Belo
Department of Informatics, University of Minho, Braga, Portugal
{rui.barbosa, obelo}@di.uminho.pt

Abstract - *Trading in financial markets is undergoing a radical transformation, one in which algorithmic methods are becoming increasingly more important. The development of intelligent agents that can act as autonomous traders seems like a logical step forward in this "algorithms arms race". In this paper we describe an infrastructure for implementing hybrid intelligent agents with the ability to trade financial instruments without requiring human supervision. We used this infrastructure, which relies heavily on artificial intelligence models and problem solving methodologies, to implement two currency trading agents. So far these agents have been able to profit from inefficiencies in the currency market, which lead us to believe our infrastructure can be of practical interest to the investment industry.*

Keywords: Algorithmic Trading, Hybrid Agents.

1 Introduction

Algorithmic trading is a form of automated trading that consists in the use of computer programs for placing trading orders, with the computer algorithm deciding on the different characteristics of the order, such as the size and the price. The adoption of algorithmic trading by participants in different financial markets is growing at an extremely fast pace. This growth is particularly noticeable in the currency market, also known as *Foreign Exchange Market* or *Forex Market*, where the use of this novel form of trading is expected to grow from 7% by the end of 2006 to 25% by 2010 [1]. The development of intelligent agents that are able to act as autonomous traders is an obvious step forward in this move away from traditional methods. With this in mind, in this paper we will describe a mechanism to implement autonomous *Forex* trading agents using an infrastructure entirely based in artificial intelligence methodologies. The idea of using artificial intelligence models in trading is not new, as there are already numerous studies in this field. Neural networks, for example, have been shown to be able to model financial time series better than traditional mathematical methods [2][3]. The use of hybrid intelligent systems for financial prediction has also been extensively researched, with several studies showing that, in general, hybrid systems can outperform non-hybrid systems [4][5].

Even though most studies seem to show that artificial intelligence models can produce reasonably accurate financial predictions, that in itself will not impress most professionals in the investment industry. That is due to the fact that these studies usually measure a model's performance based on its accuracy (for classification) or the mean squared error (for regression). From a trader's point of view, there is an obvious problem with this approach: higher accuracy does not necessarily translate into higher profit. A single losing trade can wipe out the profit of several accurately predicted trades. A low mean squared error is also not a guarantee that a model can produce profitable predictions [6]. Some studies try to overcome this problem by using model predictions on out-of-sample data to simulate trades. This is obviously a much better way to analyze the profit potential of a model, but it is still not a perfect solution. Trading simulations do not take into consideration the costs of trading, such as commissions and interest payments. They also do not take into account the problems that can occur while trading live, such as slippage and partial fills. The impact of these problems on the overall profitability of a trading strategy is not negligible.

All things considered, the performance gauges that most matter to the trading community are probably the profit and maximum drawdown. The importance given to the profit is rather obvious. The maximum drawdown, although a less known concept, is equally important. It is defined as a trader's worst period of "peak to valley" performance in the past. In other words, it is the maximum loss experienced by a trader in the past, until a new high in the profit was achieved. So the drawdown can be looked at as a measure of risk: the higher the maximum drawdown, the riskier the trading strategy will be considered.

With the needs of the trading community in mind, in this paper we will describe an infrastructure for implementing trading agents whose main goal is to maximize the profit and to minimize the drawdown while trading live. Unlike most studies in this field, which describe tools that can be used to aid the traders, we will be looking at a solution that can actually replace the traders. Therefore, each implemented agent is expected to be able to operate autonomously, placing trades and handling money management without requiring human intervention. The infrastructure is loosely based in the decision process of a traditional trader: it enables the agents to intuitively recognize patterns in financial time series, to remember previous trades and use that empirical knowledge to decide when and how much to invest, and to incorporate knowledge from trading books and trading experts into the trading decisions.

2 Forex Market

The *Forex Market* is the largest financial market in the world. In this market currencies are traded against each other, and each pair of currencies is a product that can be traded. For instance, USD/JPY is the price of the United States Dollar expressed in Japanese Yen. At the time of writing of this paper the USD/JPY price is 104.32, meaning we need 104.32 JPY to buy 1 USD. Trading this pair in the *Forex Market* is pretty straightforward: if a trader believes the USD will become more valuable compared to the JPY he buys USD/JPY lots (goes long), and if he thinks the JPY will become more valuable compared to the USD he sells USD/JPY lots (goes short). The profit/loss of each trade can be expressed in pips. A pip is the smallest change in the price of a currency pair. For the USD/JPY pair a pip corresponds to a price movement of 0.01. The actual value of each pip depends on the amount invested. For example, if we buy/sell 100,000 USD/JPY each pip is worth 1,000 JPY (100,000 times 0.01), or 9.59 USD (1,000 divided by 104.32). Even though the *Forex Market* is well known for its unpredictability, its high liquidity and volatility make it an interesting target for autonomous agents.

3 Infrastructure

An autonomous trading agent will need to make several decisions before it can place a trade. For example, it will need to decide if it should buy or sell a financial instrument, how much it should buy or sell, and when it should close an open trade. We designed an infrastructure that can be used to implement trading agents with the ability to make those decisions. This infrastructure is represented in *Figure 1*. It defines two percepts (price changes over a period of time and result of previous trades) and a single action (placement of new trades), and is composed of three interconnected modules:

- *Intuition Module* – this module is responsible for predicting if the price of a currency pair will go up or down. This prediction is done by an *Ensemble Model*, which consists of several classification and regression models that try to find hidden patterns in price data.
- *A Posteriori Knowledge Module* – this module uses information from previous trades to suggest when and how much to invest in each trade. This suggestion is done by a *Case-Based Reasoning System*. Each case in this system corresponds to a trade executed by the agent and its final result (profit or loss in pips).
- *A Priori Knowledge Module* – this module is responsible for making the final trading decision, using the prediction from the *Intuition Module* and the suggestion from the *A Posteriori Knowledge Module*. This decision is done by a *Rule-Based Expert System*, which contains several rules regarding when to invest and when to close a trade.

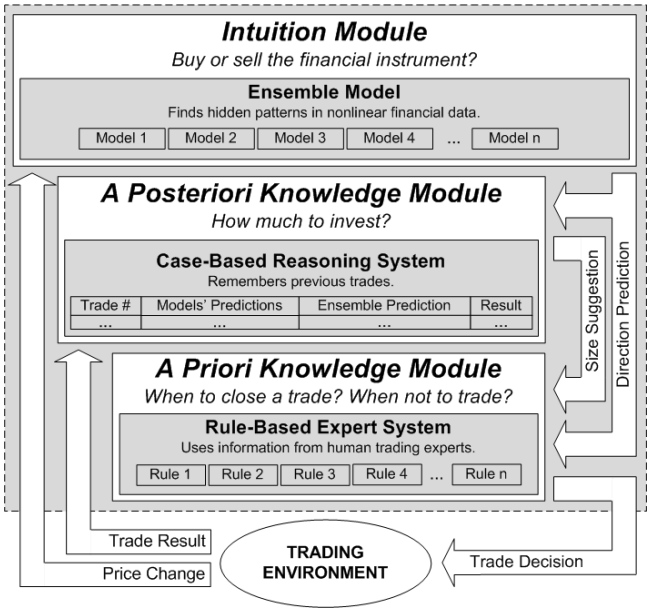

Figure 1. Infrastructure for implementing trading agents.

3.1 *Intuition Module*

In order to trade, an agent must be able to decide when it should buy or short sell a financial instrument. To do this, it needs a way to intuitively predict if the price of the financial instrument will be going up or down in the future. In a very simplistic way, we can consider that the intuitive capacity of a trader is somewhat similar to a complex pattern recognition process. With this in mind, our agents' intuitive predictions will be performed by classification and regression models capable of finding hidden patterns in financial data.

Instead of using a single classification or regression model to implement the agents' *Intuition Module*, we use an *Ensemble Model*, which should improve the prediction results [7]. This *Ensemble* is basically a weighted voting system composed of several classification and regression models, where the weight of each vote is based on the profitability of each model. The models do not try to predict the price in the future, they simply try to predict what will happen to the price in the future. The prediction of each model thus corresponds to one of two classes: "*the price will go up*" or "*the price will go down*". That is straightforward for classification models, but for regression models we need to convert the price prediction into one of the classes. That is very easy to accomplish: if the predicted price is higher than the current price then the predicted class is "*the price will go up*", otherwise if it is lower than the current price the predicted class is "*the price will go down*".

To guarantee that the agents can keep learning as time goes by, the models in the *Ensemble* will need to be retrained with new data as it becomes available. However, we only want the retrained models to become a part of the *Ensemble* if they can perform at least as well as the original models.

To accomplish this, before each prediction the *Intuition Module* splits all the available instances into two datasets: the test set, consisting of the most recent N instances, and the training set, consisting of all the instances left. The test set is first used to simulate trades with the predictions of each model in the *Ensemble* (for each test instance, if the model predicts "*the price will go up*" then a long trade is simulated, otherwise if it predicts "*the price will go down*" a short sell if simulated). The results of these simulations are utilized to calculate each model's *overall profit factor*:

$$Overall\ PF = \frac{\sum pips\ won}{\sum pips\ lost} - 1 \qquad (1)$$

Next, each model in the *Ensemble* is retrained with the training set, and the same simulations and calculations are performed with the test instances. The *Intuition Module* now has a way to select the models that will be used to make new predictions: for each model in the *Ensemble*, if the *profit factor* of the retrained model is at least as high as the *profit factor* of the original model, then the retrained model replaces the original in the *Ensemble*. Otherwise, the original is kept and the retrained model is discarded.

Once the updating of the models has been completed, the *Intuition Module* needs to decide what will be the weight of each model's vote when calculating the *Ensemble* prediction. To do so, it uses the trading simulation results obtained previously with the test data, and calculates each model's *long profit factor* and *short profit factor*:

$$Long\ PF = \frac{\sum pips\ won\ when\ predicting\ up}{\sum pips\ lost\ when\ predicting\ up} - 1 \qquad (2)$$

$$Short\ PF = \frac{\sum pips\ won\ when\ predicting\ down}{\sum pips\ lost\ when\ predicting\ down} - 1 \qquad (3)$$

The *long profit factor* measures how profitable a model was when predicting price increases in the test period, while the *short profit factor* measures how profitable it was when predicting price drops. One of these values will be the weight of the model's vote when performing a new prediction: the *long profit factor* if it predicts "*the price will go up*", or the *short profit factor* if it predicts "*the price will go down*". If the weight is a negative number the *Intuition Module* replaces it with zero, which effectively means the model's prediction will be ignored.

After all individual models have made their predictions, the *Ensemble* prediction is calculated by adding the weights of the votes of all the models that predicted "*the price will go up*", and then subtracting the weights of the votes of all the models that predicted "*the price will go down*". If the *Ensemble* prediction is greater than zero then the *Intuition Module's* final class prediction is "*the price will go up*", otherwise if it is lower than zero the final prediction is "*the price will go down*".

The purpose of the described algorithm is to allow an agent to take advantage of the different characteristics of the models in its *Ensemble*, and to make sure it can keep learning over time. Concretely, there are several reasons to use this algorithm:

- The weight of the vote of each of the individual models in the *Ensemble* is continuously updated, allowing the *Ensemble* to adapt to the market conditions. As a model becomes more profitable, its vote becomes more important.
- The *Ensemble Model* can combine the qualities of the best models at predicting when the price will go up and the best models at predicting when the price will go down. That is accomplished by using the models' *long profit factor* and *short profit factor* as their votes' weight.
- An *Ensemble Model* makes the trading strategy more resilient to changes in market dynamics. If a model in the *Ensemble* becomes unprofitable, its vote continuously loses weight, up to a point where its predictions are completely ignored. Since this model will be trained with more data, it is likely that it will end up being replaced with a more profitable retrained version of itself.
- Our algorithm optimizes profitability instead of accuracy. Obviously the learning algorithms used to retrain the models still optimize their accuracy, but the decision to actually make the retrained models a part of the *Ensemble Model* is based entirely on their profitability.
- Retraining the models before each prediction is the key to the trading agents' autonomy. The agents can keep learning even while trading, because new unseen data eventually becomes a part of the training set.

However, this algorithm has one obvious problem. It uses the simulated profitability displayed with the test data to decide if an original model should be replaced with a version of itself trained with more data. This means it selects models according to their test predictions, which might lead to selecting models that overfit the test data. But this is possibly not a very serious flaw, because models that overfit the data and are unprofitable will eventually be replaced with retrained versions of themselves (that may or may not overfit a different set of test data).

3.2 *A Posteriori Knowledge Module*

After deciding if it should buy or sell a financial instrument, the agent will need to make another equally important decision: how much should it buy or sell. In order to accomplish this, it will need a way to predict how profitable a trade will be. Using this prediction, it can optimize the profit by investing more when it expects a trade to be very profitable, and investing less when it expects it to be less profitable. One way to implement this form of money management would be to double the investment when a trade is expected to be very profitable, use a normal investment amount if a trade is expected to have moderate profit, and skipping trades that are expected to be unprofitable.

In order to determine the expected profitability of a trade, we will be looking at the individual predictions of the models in the *Ensemble*. Intuitively, we might expect that the probability of a trade being successful will be higher if all the models make the same prediction (all predict "*the price will go up*" or all predict "*the price will go down*"), compared to a trade where the models' predictions are mixed (some predict "*the price will go up*" and some predict "*the price will go down*"). Empirical evidence demonstrates that these expectations are well founded. Certain combinations of individual predictions really are more profitable than others. Our agents' money management strategy is based on that empirical observation.

We implemented the *A Posteriori Knowledge Module* using a *Case-Based Reasoning System* where each case represents a trade previously executed by the agent. The following information is contained in each case in the database: the predicted class, the trade result (profit or loss in pips) and the individual predictions from the models in the *Ensemble*. These cases are used to calculate the expected profitability of a new trade. Before the trade is placed, the *Intuition Module* makes the *Ensemble* prediction and sends the sequence of individual predictions from the models in the *Ensemble* to the *Case-Based Reasoning System*. This system retrieves from its database all the cases with the same *Ensemble* class prediction and the same sequence of individual predictions. If it cannot retrieve a predefined minimum number of cases, it removes the last prediction in the sequence of individual predictions and retrieves the cases again. This process is repeated until enough cases are retrieved, at which point the *overall profit factor* of the retrieved cases is calculated using Equation (1). That value is the expected profitability of the trade, and is used to decide how much to invest: if the expected profitability is at least as high as a predefined value the agent doubles the investment, if it is lower or equal to another predefined value it skips the trade, otherwise it uses the regular investment amount.

A new case is inserted in the *Case-Based Reasoning System* database after a trade is executed and closed, hence becoming available for the calculation of the expected profitability of future trades.

3.3 *A Priori Knowledge Module*

No matter how "smart" the implemented agents are, there is still some trading knowledge they will not be able to pick up from their empirical trading experiences. For this reason, the module responsible for making the final trading decision consists of a *Rule-Based Expert System* where important rules can be defined by trading experts.

Some of these rules can be quite simple. For example, we may want the agents to skip trades in low liquidity days, such as those around Christmas or New Year's Day. Or we may want them to skip trades whenever major economic reports are about to be released, to avoid the characteristic chaotic price movements that happen right after the release. The primary example of such a report is the United States Nonfarm Payrolls, or NFP, released on the first Friday of every month.

Other more important rules are those where the settings for take profit orders and stop loss orders are defined. These are necessary so that the agents know when to exit a trade. A take profit order is used to close a trade when it reaches a certain number of pips in profit, to guarantee that profit. A stop loss order is used to close a trade when it reaches a certain number of pips of loss, to prevent the loss from widening.

Before each trade, the *Rule-Based Expert System* receives the prediction from the *Intuition Module* and the suggested investment amount from the *A Posteriori Knowledge Module*. It then uses the rules defined by experienced traders to make the final decision regarding the trade direction, investment amount and exit conditions. The agents can be made completely autonomous by using a broker's API to send the final trade decisions directly into the market.

4 Software Implementation

We implemented the previously described infrastructure in a software program that can be easily used to create trading agents. We named it *iQuant*, short for "intelligent quantitative analyst". A screenshot is shown in *Figure 2*.

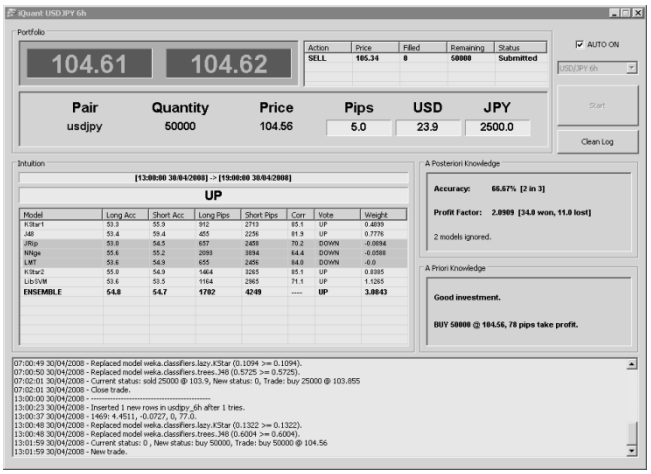

Figure 2. The *iQuant*.

The *iQuant* handles the retraining and the predictions of the models in the *Intuition Module* using the *Weka* data mining API [8]. Rules in the *A Priori Knowledge Module* are handled by the *Drools* engine [9].

Building a trading agent using the *iQuant* software involves:

- Defining the models in the *Ensemble*, and the attributes used to train them.
- Setting the parameters in the *A Posteriori Knowledge Module* to determine when a trade should be skipped or when the investment should be doubled.
- Setting the rules in the *Rule-Based Expert System*.

The *iQuant* uses the proprietary API of a broker to send orders directly into the market, but it can also be set to perform predictions without sending orders. This allows us to use it as an autonomous trader or as a tool to facilitate trading.

5 Sample Trading Agents

We used the *iQuant* software to implement two agents, one to trade the USD/JPY and the other one to trade the EUR/USD currency pair. Each of these agents places a trade every 6 hours, from Sunday 18:00 GMT to Saturday 00:00 GMT.

5.1 Agents' Intuition

To prepare the *Intuition Module* of the currency trading agents we had to select the classification and regression models to be inserted in their *Ensemble Models*. To accomplish this, we started by obtaining historical price data from a market maker's website [10]. We downloaded 4,100 candlesticks for each currency pair, comprising the period from May 2003 to January 2007. A candlestick is a figure that displays the high, low, open and close price of a financial instrument over a specific period of time. We then used those candlesticks to calculate the price return over each 6 hours period, given that this was one of the attributes we inserted in the training instances. Of the available returns, 4,000 (corresponding to the period from May 2003 to December 2006) were used to train the models and the remaining 100 (corresponding to the month of January 2007) were used to test the models.

The models were trained with attributes such as the hour, the day of the week, lagged returns and the current class or return. We also used other attributes regularly used in technical analysis by traditional traders, such as *Moving Averages* (MA), the *Relative Strength Index* (RSI) and the *Rate of Change* (ROC). *Table 1* and *Table 2* describe the attributes used to train and test each model in the agents' *Ensemble Models*. While the classification models try to predict the next class ("*the price will go up in the next 6 hours*" or "*the price will go down in the next 6 hours*"), the regression models try to predict the price return in the next 6 hours period, and that return is then converted to a class.

As previously mentioned, before each trade the *Intuition Module* uses a fixed size dataset to test the models and calculate their simulated profitability. We made the agents use a test set consisting of the most recent 100 instances, which could be considered insufficient. However, there are two reasons for using such a small set of test data:

- The shorter the test set, the faster the agents can adapt to changes in price volatility, because the weights of the models' votes are based in their simulated profitability using the test instances.
- The shorter the test set, the faster the agents can learn new patterns, because test instances will become training instances faster.

Table 1. USD/JPY agent's *Ensemble*.

Model	*Attributes*	*Predict*
Instance-Based K*	hour (nominal), day of week (nominal), MA(6), current class	class
C4.5 Decision Tree	hour (nominal), day of week (nominal), MA(6), current class	class
RIPPER Rule Learner	hour (nominal), day of week (nominal), current class	class
Naïve Bayes	hour (nominal), day of week (nominal), current return	class
Logistic Decision Tree	hour (nominal), MA(6), current class	class
Instance-Based K*	hour (nominal), day of week (nominal), MA(6), current class	return
Support Vector Machine	hour (numeric), day of week (numeric), MA(10), MA(2), current return	return

Table 2. EUR/USD agent's *Ensemble*.

Model	*Attributes*	*Predict*
Naïve Bayes	hour (nominal), day of week (nominal), current return	class
Support Vector Machine	hour (nominal), day of week (numeric), MA(2), RSI(11), ROC(12)	return
CART Decision Tree	hour (nominal), day of week (nominal), 2nd lagged return, RSI(2), ROC(2), ROC(5)	class
Support Vector Machine	hour (nominal), day of week (nominal), MA(6), MA(4), MA(3), current return	return
Least Median Squared Linear Regression	hour (nominal), day of week (nominal), 5th lagged return, 4th lagged return, 3th lagged return, 2nd lagged return, 1st lagged return, current return	return
Instance-Based K*	hour (nominal), day of week (nominal), current class	return
Gaussian Radial Basis Function Network	current return, hour (numeric), day of week (nominal), MA(12), ROC(4)	class

After completing the implementation of the *Intuition Module* of both agents, we used the modules' predictions to simulate trades with out-of-sample data corresponding to the period from February 2007 to April 2008. The accumulated profit in pips over this period is displayed in *Figure 3*.

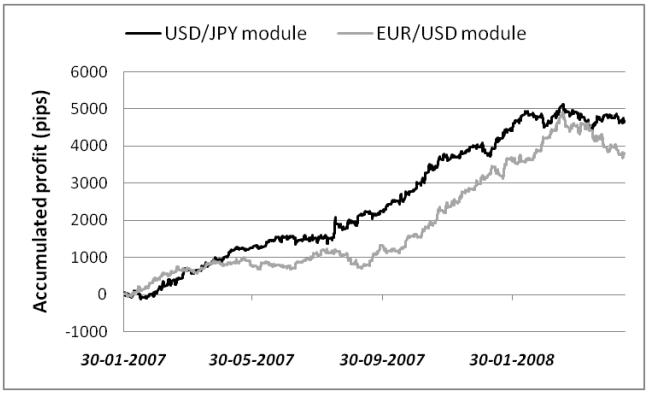

Figure 3. *Intuition Module* accumulated profit.

The total profit for the USD/JPY *Intuition Module* was 4,690 pips after 1,364 trades, with a drawdown of 671 pips. The EUR/USD profit was 3,804 pips after 1,403 trades, with a drawdown of 1,243 pips. It is fairly obvious that these trading strategies need improvement: the drawdown is too high and the profit curves are too erratic.

5.2 Agents' Empirical Knowledge

To prepare the agents' *A Posteriori Knowledge Module* we just needed to define a couple of user variables regarding money management. After a couple of trial and error tests, we decided to use the following settings:

- double the trade size if the *overall profit factor* of the cases retrieved is greater or equal to 0.5;
- skip the trade if the *overall profit factor* of the cases retrieved is lower or equal to 0;
- require a minimum of 3 retrieved cases before a decision is taken.

Figure 4 shows the result of combining the *Intuition Module* and the *A Posteriori Knowledge Module* of each agent to simulate trades using the out-of-sample data. The USD/JPY combination performed 996 trades, with a final profit of 8,897 pips and a drawdown of 1,103 pips. The EUR/USD combination performed 1,031 trades, with a profit of 4,932 pips and a drawdown of 1,473 pips. Even though there was a great increase in the profit and a decrease in the number of trades, these strategies still need improvement because their drawdowns are too high.

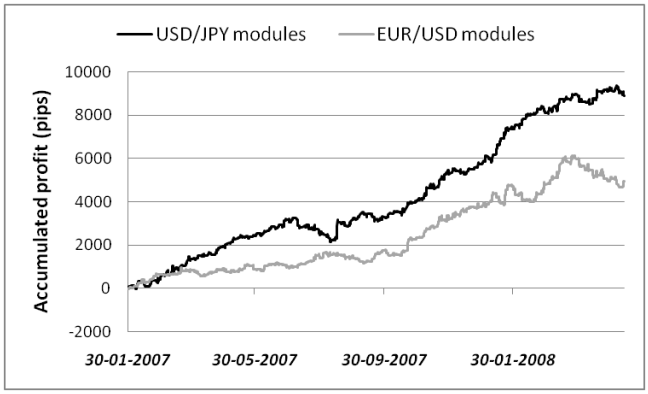

Figure 4. *Intuition Module* and *A Posteriori Knowledge Module* accumulated profit.

5.3 Agents' Expert Knowledge

To prepare the agents' *A Priori Knowledge Module* we just had to define the rules in their *Rule-Based Expert Systems*. We inserted rules to stop trading in the following low liquidity days: Christmas, New Year's and Good Friday. We also added a take profit rule for each agent. The USD/JPY agent was set to automatically close a trade if the price moved 0.40% in the predicted direction, while the EUR/USD was set to do the same if the price moved 0.20%. Since we did not insert any other rules, a trade that was not closed with the take profit order would only be closed when the 6 hours period ended and a new trade was opened.

Figure 5 shows the results of combining each agents' *Intuition Module* and *A Priori Knowledge Module* to simulate trades with the out-of-sample data. The USD/JPY combination netted 4,772 pips of profit after 1,325 trades, with a drawdown of 465 pips. The EUR/USD combination obtained 4,416 pips after 1,363 trades, with a drawdown of 624 pips. These lower drawdowns are exactly what we required, but unfortunately there is also a big profit reduction if we compare these results with the ones obtained with the combinations between the *Intuition Module* and the *A Posteriori Knowledge Module*.

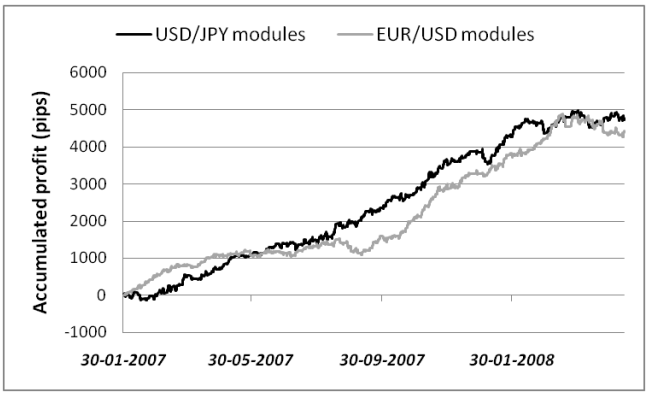

Figure 5. *Intuition Module* and *A Priori Knowledge Module* accumulated profit.

5.4 Results

Through simulation, we have shown that each module makes a different contribution to the profit and the drawdown. The actual agents, implemented using the *iQuant* software, consist of all the three modules working together. *Figure 6* shows the simulated trading results for both agents. They were able to add the *A Posteriori Knowledge Module* ability to increase the profits to the *A Priori Knowledge Module* ability to reduce the drawdown. The USD/JPY agent obtained 8,308 pips of profit after 971 trades, with a drawdown of 760 pips. The EUR/USD agent obtained 5,944 pips after 1,005 trades, with a drawdown of 798 pips.

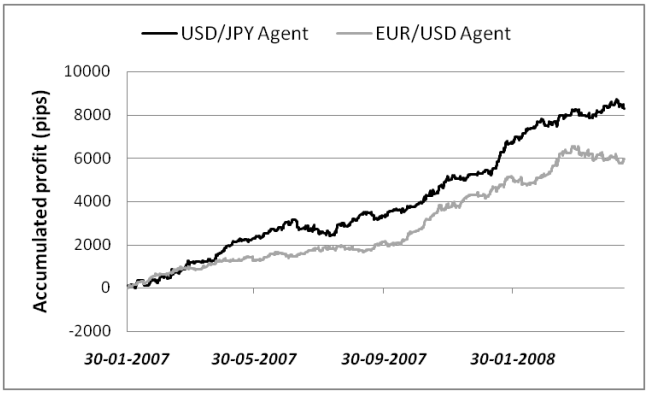

Figure 6. Trading agents' accumulated profit.

Table 3 resumes the trading statistics for both agents and the combinations of their modules. It is interesting to notice that the *Intuition Module* of the USD/JPY agent can only predict if the price will go up or down with 53.59% accuracy, while the EUR/USD module can only do it with 52.82% accuracy. These percentages might seem too low, but we must keep in mind that these modules optimize profitability instead of accuracy. Therefore, even though they are not very accurate, the profit they obtain from accurately predicted trades is a lot higher than the losses suffered from incorrectly predicted trades. The modules' success rate, i.e., the percentage of trades that are closed in profit, is equal to their accuracy because all the trades are closed at the end of the 6 hours period, when a new trade is opened.

Another interesting statistic in this table is the low number of trades performed by the agents, when compared with the number of trades that would be performed by their *Intuition Modules* alone. That difference is due to the fact that the agents' *A Posterior Knowledge Module* and *A Priori Knowledge Module* can make them skip trades that are expected to be unprofitable. Also notice that both agents have a success rate that is considerably higher than their accuracy. That is due to the take profit rule in their *A Priori Knowledge Module*. This rule allows the agents to be profitable even if they make a wrong prediction.

Table 3. Trading statistics.

	Modules	*Acc. (%)*	*Succ. (%)*	*Profit (pips)*	*DD (pips)*	*Trades*
USD/JPY	Intuition	53.59	53.59	4,690	671	1,364
	Intuition + APosteriori	56.33	56.33	8,897	1,103	996
	Intuition + APriori	53.43	55.02	4,772	465	1,325
	Agent	56.33	57.67	8,308	760	971
EUR/USD	Intuition	52.82	52.82	3,804	1,243	1,403
	Intuition + APosteriori	53.64	53.64	4,932	1,473	1,031
	Intuition + APriori	53.19	57.96	4,416	624	1,363
	Agent	54.03	59.00	5,944	798	1,005

To make the analysis of the results more intuitive, it might be interesting to see how the agents' performance would translate into actual money won or lost. *Forex* investments are usually leveraged (which means they are done with borrowed funds), so the total profit obtained by an agent will always depend on the leverage it uses. Let us assume we had a starting capital of $100,000, and using the rules in the *A Priori Knowledge Module* we instructed our USD/JPY agent to use a maximum 2:1 initial leverage, by setting its trade size to 100,000 USD/JPY. In this scenario, the agent only uses the maximum leverage when it doubles the trade size for trades with high expected profitability. For a USD/JPY price of 104.32, the pip value for a 100,000 USD/JPY trade is $9.59. Since our agent obtained a total profit of 8,308 pips, its profit in dollars after 15 months of trading would be $79,674, or 79.7%, and its maximum drawdown would be $7,288. If the same scenario is applied to the EUR/USD agent, it would use a 64,000 EUR/USD trade size (100,000 divided by 1.5625, which is the current price of one EUR in USD), and each pip would be worth $6.40 (64,000 times 0.0001). Therefore, its profit would be $38,042, or 38.0%, and its maximum drawdown would be $5,107.

As previously mentioned, simulated results can give us a general idea regarding an agent's ability to be profitable while trading live, but does not provide any guarantees. The only way to prove that an agent can actually be profitable is to let it create an extensive track record of live trading. In order to accomplish this, our *iQuant* agents have been trading autonomously for several months using an *Electronic Communication Network*. As expected, the agents' live trading results are not as good as the simulated results, with a decrease of around 25% in the total profit. This difference is due to commissions, slippage, partial fills and interest payments, amongst other things. However, these results are still very satisfactory. Even though it might be too soon to reach any conclusions, they seem to demonstrate the agents can profit from inefficiencies in the *Forex Market*.

In order to reduce the trading risk, we could try to diversify the investment by using the *iQuant* software to implement agents with the ability to trade other financial instruments, such as stocks or futures. We are currently looking into this multi-agent investment strategy, which we believe could be of practical interest to the algorithmic trading community.

6 References

[1] Toby Cole. "Foreign Exchange Implications of Algorithmic Trading". Foreign Exchange Contact Group, May 2007.

[2] Christian Dunis and Mark Williams. "Modelling and Trading the EUR/USD Exchange Rate: Do Neural Network Models Perform Better?". Derivatives Use, Trading & Regulation, No. 8/3, pp. 211–239, 2002.

[3] Joarder Kamruzzamana and Ruhul Sarker. "Comparing ANN Based Models with ARIMA for Prediction of Forex Rates". ASOR Bulletin, Vol. 22, No. 2, pp. 2–11, 2003.

[4] Ajith Abraham. "Analysis of Hybrid Soft and Hard Computing Techniques for Forex Monitoring Systems". Proceedings of the 2002 IEEE International Conference on Fuzzy Systems, pp. 1616–1622, 2002.

[5] Lean Yu, Shouyang Wang and Kin Lai. "Designing a Hybrid AI System as a Forex Trading Decision Support Tool". Proceedings of the 17th IEEE International Conference on Tools with Artificial Intelligence, pp. 89–93, 2005.

[6] Kevin Swingler. "Financial Prediction, Some Pointers, Pitfalls, and Common Errors". Neural Computing & Applications, Vol. 4, No. 4, pp. 192–197, 1996.

[7] Pádraig Cunningham. "Ensemble Techniques". Technical Report UCD-CSI-2007-5, April 2007.

[8] Weka API, http://www.cs.waikato.ac.nz/ml/weka/.

[9] JBoss Drools, http://www.jboss.org/drools/.

[10] OANDA Corporation, http://fxlabs.oanda.com.