

CS404: Agent based Systems

Trading Agents

Matthew Cranham, Michele Mancina, Jessica Nickson,
Faiz Sayyid and Robert Steele

Jan 2012

1 Introduction - Auctions

Wooldridge et al.[1] define an auction between agents as a mechanism to allocate scarce resources to agents. The term resource is here used in its widest sense, for example, a painting might be considered a resource as much as the bandwidth available in a network. The scarcity of the resource is what makes the need for an auction arise, as if there is enough of the resource for every agent, then the allocation is not a problem.

One of the most important features of auctions is the efficiency in allocating the resource, meaning that through an auction, these assets will be allocated to the agents that need them the most. There are different types of auctions, but in any auction between agents there will be an agent, called the auctioneer, in charge of managing all the bids received and assigning the resources to the agents, and a collection of agents called the bidders, which are the actual participants to the auction. In every auction, no matter what type of auction, it will be in the auctioneer's interests to maximise the profit from allocating the resource, and it will be in the bidder's interests to find the best strategy to minimise the expenses while still acquiring the desired resources.

There are different attributes that need to be considered when designing an auction mechanism:

- common/private/correlated value: this is the value that agents give to goods, in the case of common value, the value will be the same for every agent; in the case of private value, each agent will value the goods differently; finally in the case of correlated value, each agent will value the goods in relation to how the other agents value those same goods.
- first/second price: this attribute defines whether the winner of the auction will have to pay the actual bid (first price auction) or the amount of the second highest bid (second price auction).

- open cry/sealed bid: this attribute dictates whether each bidder's bids are publicly announced or kept secret from every other agent.
- ascending/descending auction: this final attribute makes a distinction between auctions where the starting reserve price (the minimum selling price) is announced and the bids going in ascending order until there are no more bids and auctions where the auctioneer starts by announcing a price and going down until one or more of the bidders are willing to pay that price.

The final distinction that can be done is between auctions for single items or auctions for multiple items at the same time, where bidders place bids on bundle of goods.

There are different types of known auction mechanism, such as English auctions, Dutch auctions or Vickrey auctions. Vickrey auctions, being second-price, sealed bids type of auctions on single items, are very similar to the auction setting for the TAC/AA competition, with the exception of a few aspects, such as the squashing factor or the fact that in the TAC/AA competition, agents place bids over bundles of goods. Wooldridge et al.[1] point out that the most important feature of Vickrey auctions is that they make bidders place bids for the amount they actually value the items, that is, truth bidding is the dominant strategy for the bidding agents.

When considering the TAC/AA competition however, it's not that obvious what is the true value that each query holds for the bidders and it's still unclear what the dominant strategy should be. For these reasons different approaches have been researched and experimented, experiencing different degrees of success. Here we list and discuss these different approaches.

2 Game Theoretic Approaches

Game theory is best described as a variant of decision theory, where the result of taking a decision depends on the actions of another player. Usually such an opponent is trying to maximise their own benefit at the cost of the first player. Games normally consist of two players choosing a move from a limited set of moves. The payoff that a particular player receives depends on the actions (moves) of both the players. This dictates that, in trying to achieve maximum payoff, one must try to predict the most likely course of action that the other player will take. Game theory has been proposed as a framework for analysing these games. The first mention of game theory being applied to interacting agent systems was in 1985 [3].

Parsons and Wooldridge[2] point out that previous studies on game theory for multi-agent systems typically assumed that agents can select the best course of action in the space of all possible strategies, considering all possible interactions. However, in real world situations, this assumption is unrealistic. The

main issue is that the search space of all possible strategies and interactions grows exponentially, hence making the search of the optimal strategy computationally intractable.

In this light the concept of bounded rationality becomes of crucial importance. This concept arose from the work of Simon[4][5][6], who argued that people, not having unlimited processing power at their disposal, settle for satisfactory rather than optimal solutions, that is, solutions that are good enough. For these reasons, as suggested by Stirling[7], realistic implementations of game theoretic agents should focus on the trade-off between the benefits and the computational cost of the solution rather than just the benefits. Stirling[7] discusses a new way to explicitly model the cost in resources and to balance it against the optimality of the solution, thus introducing the notion of praxeic utility.

In the past, game theory has only been used quite rarely in designing auction agents, for a number of reasons. In their paper, Vorobeychik et al.[8] try to identify some of the main reasons for the difficulty in implementing game theory derived agents. First of all, in such a complex environment as keyword auctions, there might be many equilibria. Secondly, equilibrium selection requires coordination between agents and in a game with so many competing agents, where coordination cannot be assumed, this is a non trivial problem. A third, more important, issue is that some among the competing agents could have irrational behaviours, hence rationality cannot be assumed either. The reasons for this are many fold; they may be buggy, or the designers of the other agents may have purposefully designed theirs in such a way as to be irrational in order to hamper the performance of any game theoretic derived agents. These considerations can close doors on the ability of one to operationalise a game theoretic agent. Vorobeych et al.[8] attempts to guide agent designers in this process of operationalisation with regard to the concerns voiced previously.

A more intuitive approach to multi-agent bidding environments, and indeed the most common, is the use of optimisation and machine learning concepts. Tactex05[10] and Tactex09[9] are both extremely successful examples of agents based upon machine learning concepts. The authors of these papers utilise these techniques in the design and implementation of successful, competition level agents.

On the other hand, Vorobeychik et al. suggest that machine learning has limits and that, in the instance that information about its adversaries is unavailable, an agent may perform poorly and be unable to make a good decision.[8]

Vorobeychik[8] also points out that, if these other adversaries are learning at the same time, then complex interactions will occur and these interactions will likely interfere with assumptions made in machine learning algorithms. Vorobeychik[8] admits that machine learning has its place, but that they are only testing the utility of a game theoretic agent. An obvious expansion of their work would be

to develop a hybrid agent that is capable of both forms of reasoning. This agent would be able to use game theory when certain criteria were reached (described above).

It is important to note that game theoretic approaches are not a panacea in all circumstances, as Pardoe and Stone[11] point out. They claim that game theoretic approaches are of little value when one has incomplete information about the other players in game. Pardoe and Stone[11] attempt to lift us out of the dogma that dominates agent design; that they can be either game theoretic or utilise machine learning in style. It is their view that in situations where one has not enough time to learn about the environment and where not enough is known about one's adversaries, the agent should seek to employ past knowledge in order to dominate. Pardoe and Stone's[11] paper deals with how an agent should apply past knowledge referring to the re-application of knowledge as transfer learning. Past champion trading agents such as Tactex09[9] design their agents strategy around price prediction that is accomplished by using training regressions models using supervised learning algorithms. There has been little research done, however, on how to fuse transfer learning and regression algorithms. Research in this area could provide a huge break through in agent performance when an agent enters an entirely new market.

2.1 A game theoretical approach to the TAC/AA competition

Vorobeychik et al.[8] describe an agent based on game theoretic analysis that participated to the TAC/AA competition with excellent results, reaching the stage of the finals. The most impressive aspect of the agent is definitely its simplicity, especially when compared to other agents like TACTex2009 (see section TACTex2009).

2.1.1 Bidding policy

The first problem faced is that of simplifying the complex competition setting into a more computationally tractable one. This is accomplished following the advice of Vorobeychik[12], that discusses and justifies the use of a simple linear bidding strategy of the form of $b(v) = \alpha v$, because it is used by the advertiser to decide what fraction of their real value they want to bid.

v represents the value per click for the advertiser and α , the 'shading factor', is in the range $[0,1]$.

This is, of course, a very strong simplification, referred to as myopic by the author themselves, because it doesn't take into account any state information available throughout the game.

2.1.2 Estimating value per click

Once the authors have established a simple model for bidding policies, the whole problem lies in the estimation of the value per click function. Intuitively, the value per click of a keyword q for an advertiser a is the expected earning for each click on the ad. That is, the probability of conversion (given that the user already clicked on the ad) multiplied for the revenue in case of conversion. In order to compute the probability of conversion, different variables need to be estimated, such as the proportion of focused shoppers (that is, shoppers that are interested in buying), the CTR (the click-through rate) and the total amount of impressions throughout one day. Most of these estimates are obtained through the execution of several offline simulations and calculating the average of the values empirically observed in these simulations.

2.1.3 Game Theoretic Analysis

At this point the space of all the possible strategies is restricted to what value to give to α . In order to do so, Vorobeychik et al.[12] make a further simplifying assumption: that each other agent in the competition uses the same bidding policy, hence focusing on what is called symmetric strategy profiles. On one hand this assumption is needed, because it's much harder to operationalise an asymmetric equilibrium (and it is unclear how to assign different roles to different agents) and on the other hand there is not necessarily much loss from the descriptive point of view, because what the agent ultimately cares about is just what the other agents do in the aggregate, as their strategies influence parameters such as the focused user proportion or the CTR.

Furthermore, the space of all possible values that α can take is discretised to the first decimal value, so $[0.1, 0.2, \dots, 1.0]$ with the exclusion of the value 0, because every bid has to be strictly positive.

Through the use of an iterative best response approach, the authors finally find a convergence in the best strategy adoptable, that is, what value to give to α .

3 Machine Learning Approaches

Many techniques may be used for the implementation of a machine learning process e.g. Classifier Systems, particle filters, Genetic Algorithms, Neural Networks, and mechanisms of reinforcement learning [13].

Stone et al.[14] have compiled a review of machine learning techniques that they feel are most compatible with agent design. These include:

1. Simply bidding the current price, not a machine learning technique but certainly a component that could be used a machine learning algorithm.
2. Compute the averages of the differences between the closing price and current price based upon prior games. Inform your prediction with this data.
3. Use a curve fitting algorithm such as Levenberg–Marquardt to the current sequence of pricing information to extrapolate a winning bid.
4. Use the information about the closing bid prices from previous games to predict today's closing price.
5. Learn a mapping of game features to closing price.

ATTac-2001 takes approach five, this approach is usually taken when machine learning is used. This list is far too general and out of date. Point 5 includes a plethora of advanced machine learning techniques and is therefore far too general. Advanced Machine learning techniques have existed since 2001 and so it is strange that more explicit definitions were not included. Indeed it was impossible to find an up to date review of learning techniques in bid price prediction. There exist many papers on stock price prediction using machine learning, however, these could not always be used as the theory upon which they were based drew upon economic theory relating to stock price behaviour that was in no way applicable to bidding.

Furthermore the odd nature of the bidding system is different to traditional auction theory and so introductory/accessible books and papers on the subject were of no use. Therefore an attempt to discuss common machine learning techniques applied to agent based systems is shown in the next sections.

3.1 Neural nets (NN)

Whilst neural nets are useful in machine learning in order to solve any problems that are reasonably complex, multi-layered neural nets need to be used. This is because perceptrons are limited in the type of concepts they can learn, as they can only learn linearly separable functions. For example, one cannot simulate a XOR operation with a NN. These multi-layered nets are distinct in that they use hidden layers, named as such because of the fact the outside world does not communicate with these components. [15]

Back propagation is the learning algorithm for a multi-layered artificial neural network (MLANN). It is derived using differential calculus that relies on

having a continuous threshold function. Perceptrons, having a non continuous threshold function, do not lend themselves to back propagation techniques. Sigmoid units are one of the alternatives used.

When using MLANN in the real world on real agents, one has to exercise restraint. There are limitations to the types of problems MLANNs can solve. One dimension to consider is the problem of over training. Left unsupervised, back propagation training in MLANNs can result in the network beginning to orient itself to individual aspects of the data, rather than the data set as a whole. This can lead to the multi-layer network over-fitting itself to the training examples.

The over-fitting problem can sometimes be rectified by letting the network continue to train. However, this behaviour is not suitable in a real world agent, as the agent would perform sub-optimally while it trained itself and would therefore be completely unsuitable for short games - as is the case in with the TAC/AA competition.

3.1.1 Appropriate Problems for Artificial Neural Network (ANN) Learning

When using ANNs for learning, a fair amount of time must be provided and long training times need to be acceptable. Training times can vary from a few minutes to a few hours. This is clearly unacceptable for our agent as it can only learn in a live environment and a game only lasts 60 seconds. Secondly, ANNs are black boxes and it is difficult for a human outsider to determine exactly what is going on inside. This would not be a problem in our situation. Thirdly, once trained, ANNs are very quick. They have been used in battlespace environments when the need for a fast and accurate result is paramount. This speed is dependant on them having been trained using a large set of data, for a long period of time. An ad auction agent used in a commercial setting could be trained for a long time by having the agent simply monitor proceedings in the auctions and not letting the agent actually make any purchases. Once trained, the agent could then be allowed to start bidding.

However, this model is not compatible with the way that the TAC works and so it is likely that ANNs would be of limited value unless a large training set could be found and the agent trained before use.

3.2 Q-learning

Q-Learning is one of the simplest algorithms which implements Reinforcement Learning (RL), a technique that allows software agents to automatically de-

termine the best course of action in a specific environment state to achieve maximum payoff. A simple feedback technique, called a 'reinforcement signal', is used to learn the behaviours.[16]

Reinforcement learning research faces several challenges. To begin with, it is simply too expensive to store values of each state, with regards to the memory requirements. This limits RL to simpler problems. This limitation can be solved by approximation techniques, but these are seldom perfect. Most importantly, due to limited perception of the environment, fully determining the current state is often quite tricky.

In Q-Learning, as discussed, the agent learns a mapping that dictates the next course of action when the environment is in a certain state. This mapping is best described as a table, the rows of which represent all the states the environment could possibly be in and the columns are the actions the agent can take[16]. The value of a particular cell in the table shows how favourable taking the corresponding action would be. Sometimes, the agent can use its experience to determine which actions are the best to take. The only metric an RL agent needs is the reward signal, which tells the agent whether or not the last action was good or not.

For each action taken, the agent earns a reward proportional to its performance. For example, the pole balancing problem is a standard test for Reinforcement Learning algorithms. One might implement a reward scheme where the reward given to the agent could be positive while the pole is standing and, when the pole is dropped, the reward changes to a negative number - "punishing" the agent. [17]

Q-learning's greatest boon is its simplicity. It is easy to implement and garners good results in a variety of applications. There are, however, constraints to when Q-Learning may be employed. The first and largest problem is that the number of possible environment states and actions the agent can take has to be finite. As the number of states and actions increase, the Q-table and search space become huge and performance decreases dramatically. With real life problems, the number of possible states and actions is often intractably large, but when confined to our TAC/AA game -an artificial environment- it may be feasible to implement part of the agent as a RL device. The other danger is getting marooned at a local maxima. Taking the highest Q value for each state may result in better actions being ignored, as the agent always prefers things that "worked well before" and will never try for better actions. This can be mitigated by using an exploration strategy.

The SARSA algorithm is a variation of Q-learning. SARSA's learns depending on the action taken by the agent; the agent's decision policy. This is also known as the exploration factor. On the other hand, in Q learning we do not care about the action that was taken, rather what the optimal action to take would have

been. This is the only difference between the two techniques. Given sufficient training Q learning will always converge to a very close approximation of the function being learned. SARSA, on the other hand does not necessarily converge.

He and Jennings[18] show that an agent's performance is dependant upon the attitudes to risk that its rivals possess. They discover that risk-averse agents are highly flexible and thrive when there is a great deal of competition in a market. A risk-seeking agent, however, seems to perform better in a non-competitive environment. This research has a great deal of value attached to it and has been used to design subsequent trading agents. It is therefore fitting that any future agents be able to tailor their attack strategy to the current market environment; seeking risk when there is little competition and being conservative when prices are diverse.

3.3 Genetic Algorithms

Genetic algorithms(GA) mimic the process of natural selection in order to heuristically generate solutions to optimisation and search problems. The first step in using a genetic algorithm to solve a problem is to find a way of encoding the problem; this is usually accomplished by representing the problem as a bit string. This bit string is referred to as a chromosome. At the beginning of a particular run of the algorithm a large quantity N , of random chromosomes are created, with each representing a possible solution to the problem.

A general GA will follow the following steps:

1. For each chromosome test how effective it is at solving the problem and assign it a corresponding fitness score.
2. Select two members of the population, with the probability of being chosen proportional to the fitness of the solution. For this step roulette wheel selection is commonly used
3. Crossover the bits from each chosen chromosome at a randomly point dependant on a variable defined as the cross over rate.
4. Iterate through the new chromosome and mutate it by randomly flipping bits-dependant on the mutation rate
5. Repeat 1) to 4) until a population of N new chromosomes have been created.

The mutation rate is typically low 0.001 for a binary chromosome.

Several criticisms are levelled at the genetic algorithm when it is compared to alternative optimisation routines. Step 1) in which a fitness function is repeatedly evaluated is often the most computationally expensive step in the entire process,

especially when finding optimal solutions to high dimensional problems. The more difficult the problem, the more complex the fitness function. This can be mitigated to a certain degree by using appropriate approximations to the fitness function.

In many cases GA tend to converge at a local optimum rather than a true global optimum, as some GA are myopic in that they cannot realise that occasionally short term fitness must be sacrificed to gain longer term fitness. This unfortunate circumstance may be rectified by increasing the rate of mutation and in doing so maintain a population of solutions that show great variation. Other variation assurance techniques are used such as imposing a niche penalty where groups of chromosomes that are too similar are removed from the pool and even solutions where completely random chromosomes are periodically injected into the population.

Operating on dynamic data sets, as is the case with an ad auction agent, increases the difficulty even more as populations of chromosomes begin to converge on a solution which may no longer be valid several trading days into the problem.

In the work of Munsey et al.[19], the fitness function of a chromosome is defined as the average profit earned by a chromosome in the game. This requires two runs of the game, making the fitness function time and computationally expensive as identified earlier.

The initial chromosome population is set to 16 with the fittest chromosome automatically being chosen for the next phase of the game, in addition to 15 other pairs of chromosomes chosen for reproduction.

Munsey et al.[19] maintain variation in the population by giving a high value for mutation (0.02) in addition to other randomisation tricks. The time taken to generate one new population is dependent on the run time of a game. This is problematic as the authors suggest it takes approximately 13 hours to start performing well.

3.4 Particle Filter - TacTex 09

A discussion of the TAC AA competition would not be complete without a discussion of the champions. TacTex09 can be viewed as being a machine learning agent as at its core is a particle filter. TacTex's methodologies due to their unbridled success enjoy a great deal of analysis in other academic works and so it is only fitting that we cover it comprehensively here - indeed more so than any other paper. In addition to this our team took the decision to trial a particle filter dependant agent.

As the winner of the inaugural TAC/AA competition TacTex09[9] is one of

the most cited papers in the field of designing an ad auction agent. This is partly due to its success and partly due to the fact that, after this competition, details of the subsequent winners are exceedingly hard to come by. This attitude is exemplified by the change of heart of the entrants to remove the source code from the TAC/AA website and instead give only binaries. Despite the fact that progress will have been made in the years between 2009 and the present day, none of this progress can be either used or discussed due to the fact that it is shrouded in secrecy. Therefore for all practical purposes Tactex09 is the state of the art. The paper's authors give both high and low level details of the agent, with particular attention paid to the optimisation component that places the bids. In general, the agent performs its task by inferring the value of certain variables and using these inferences to best propose a bid.

Tactex's construction is modular and so is the technical report; it therefore makes sense to break down the analysis of the agent by the different operating modules.

3.4.1 Position Analyser

This is the first stage on the agent's production line; a pre-processor that analyses the previous day's sales reports and converts them into a form that enables the ranking of the quashed bids to be deduced. The first and last impression for each advertiser can be used to figure out who has reached their bidding limit. The system of equations produced cannot be solved exactly, so a search algorithm is employed. Ninety-nine percent of the time this algorithm returns results for the position of a rival agent. The position is guessed from a number of heuristics and, typically, the best solution to the series of equations is chosen.

3.4.2 User Model

Each user is interested in a specific product-brand combination, which means that there are nine different user types and it is the job of the user model to determine the sizes of the of the different population states. This is accomplished by constructing a particle filter for each of the nine types of user populations, which consists of 1000 particles representing 10,000 users. The particle filter takes the form of a sequential Monte Carlo method. Each day, a new set of particles is generated from the old. This is done by selecting a random old particle according to its weight and the new particle's user distribution is generated.

To predict the user population n days into the future we update each particle $n+1$ times as per the transition matrix. This can be done because we now have the distribution over the population state on day $n-1$.

3.4.3 Advertiser Model

This module is concerned with the generation of three types of prediction; impression predictions, ad predictions and bid estimations. The impressions predictions section tries to predict the maximum number of impressions that each advertiser could possibly make before it reaches its budget. TacTex09 does this by asserting that prior behaviour is a good indication of future behaviour and therefore assumes it will be the same as on the previous day, or that no spending limit exists. The ad predictor tries to determine which type of ad – targeted or generic – will be used by each individual rival advertiser. This is done by counting all the ad types seen so far in the game for each advertiser per query. The predicted ad for a query is the most common one. However, bid estimation is the hardest problem, because the bidding behaviours of the other agents are unknown. TacTex09 uses two discrete techniques that rely on totally different ideas. Neither technique was shown to conclusively outrank the other, but averaging the output of both produced superior results. The first models all competitors’ bids jointly, whilst the second technique models bids separately.

Method 1 uses another particle filter technique to estimate rivals’ bids, using one filter per query type and having one particle representing a set of rivals bids for that query type. Associated with each set of filter particles is a probability distribution that gives the likelihood of each particle representing the actual current state. As before, the distribution is sampled from each day to obtain the next set of particles. Each particle is then updated based upon the days’ observations (CPC and rankings). TacTex09 corrects a single bid of one of the advertisers, on the condition that it is needed, so it is consistent with the rankings and CPC. The probability distribution is then re-computed for the set of new, adjusted particles.

In the second approach, the agent maintains separate bid distributions for each advertiser and it approximates the distribution by using a fixed set of bids. The bid space is discretised using an exponential function, which is believed to achieve a better prediction of low bids. These bids are the most common according to their research. The authors assume bids change in one of three ways: ‘randomly’, which is difficult to model, ‘only slightly’ - with probability proportional to the density function of the zero mean normal distribution and finally, bids changing with respect to bids made 5 days ago. Bids often follow 5 day cycles, which is imposed by the 5 day capacity window. The probability of each bid can then be found.

3.4.4 Parameter model

This section predicts two things: 1) the probability that a user will progress from one advert to the next, γ_q . 2) a probability that effects a user clicking its advert. The prediction is calculated by maintaining a joint distribution over the

two parameters.

3.4.5 Optimisation high level

The previous modules are concerned with the estimation of the game state and predictions of the future. It is the job of the optimiser to use these results to make decisions. The most important part of choosing bids, ads and spending limits for each query is the distribution constraint. TacTex reasons only about conversions and acts to bring about those conversions. This optimisation consists of three levels that operate in a trickle down fashion. At the first level is the Multi-Day Optimiser (MDO). As the objective is to win the entire game, not just each day, it is the MDO's responsibility to determine how many conversions to aim for per day. The second level Single-DO decides how to divide conversions among the 16 query types according to a single day's target. In addition, it computes the expected profit. It is the Query analyser's job that determines the tuple of bid, ad and spending limit that will bring about the number of conversions the SDO wants.

3.4.6 Optimisation low level

The query analyser simplifies its job by deferring the choice of spending limits and instead, focuses on controlling the conversions using the bid that is more profitable. For a given bid, each ad is evaluated and one is picked based on which will give the highest profit per conversion. The spending limit is set equal to the expected cost. There is the additional challenge that a competitor's bid may fall between two predicted discretised positions and thus the function must be performing linear interpolation between pairs of points. In summary:

1. Compute predicted conversions, costs and revenues for each of the eight bids
2. Interpolate to establish a function that maps between bids and conversions and costs
3. For a given conversion target find the bid that gives this target from the mapping in step two. In addition determine its cost and expected revenue from the other precomputed mappings.

3.4.7 The SDO

Using the query information, the SDO can determine the optimal number of conversions to target for each query type, for a daily target. Pardoe et al.[9] chose a nearly-optimal greedy algorithm to solve this problem. Using a dynamic programming technique for solving a multiple choice knapsack problem would have resulted in slightly better results, but the time complexity of this technique was far greater and did not justify its use. Adding conversions from the most

profitable query types forms the basis of the simpler algorithm. The higher position is not worth paying for unless you are making greater than a critical number of conversions.

3.4.8 The MDO

To maximise profit over the entire game, the actions that must be taken on all days up to the game's end must be considered. The MDO's role is to find the optimal set of conversion targets for the remainder of the game. By running the SDO on each remaining game day, the expected profit from any set of conversion targets can be found.

Formally the MDO's goal on a day d , is to find conversion targets. Planning for the entire game can be very computationally expensive if an optimum solution is to be found - again using dynamic programming. Instead, the MDO uses a hill climbing search to find a solution. Despite requiring fewer resources, the hill climbing algorithm performed better.

3.4.9 TacTex Conclusion

In the 2009 TAC/AA competition, TacTex not only placed first but it also scored significantly higher than its nearest competitors. From analysis of the tactex09 technical report the following can be gleaned:

1. As a consequence of the capacity window constraint, TacTex's performance varies. Importantly, at the end of the game where the capacity constraint no longer applies, TacTex performs best.
2. TacTex's predictions become more accurate as the competitor advertisers' behaviour converges.
3. The bid estimators perform well - any change that the authors tried resulted in a negative impact on agent performance.
4. The MDO is one of the most important components in the agent.
5. Choosing a targeted ad is simple strategy with a good return.

3.5 Machine Learning Conclusion

It can be seen from the review of various techniques of machine learning that they are powerful techniques that enjoy a good deal of exposure to the TAC AA environment.

TacTex09[9] for example, enjoyed a great deal of success using its particle filter.

Machine learning does have its difficulties however; the more powerful techniques, for example, particle filters are difficult to implement.

Given that only a finite amount of time is available it would make sense to not rely solely on this approach. In our case it was decided that multiple agents would be worked on. One that used a machine learning approach as in TacTex09[9], a simple adaptive approach and one that featured a simpler machine learning technique; a genetic algorithm approach. The ability to adapt is the most challenging component of the agent and the genetic algorithm approach seems to offer the best adaptability to complexity ratio.

As pointed out earlier, the most challenging aspect of a GA is the fitness function. Given the numerical nature of the problem this should not be as difficult to implement if this method were chosen. Other techniques, whilst a great deal more simple (e.g. Q learning), would become intractable due to the size of the problem. No papers could be found that used Sarsa learning and whilst it would be interesting to experiment with this technique, the literature seems to indicate that it is extremely time consuming with an unknown pay off value and therefore seems unsuitable for this piece of work.

4 The State of the Art

4.1 A First Approach to Autonomous Bidding in Ad Auctions

Greenwald et al.[20] take a different approach to develop an autonomous bidding agent. The authors first create an agent for a stylized problem and then adapt the agent for the TAC/AA game. The authors concentrate less on prediction of the scenario and more upon the optimization problem. Their approach to the optimisation problem is far different to any other. It consists of two techniques: A rule based approach that can make reasonable decisions, even with inaccurate data, and a greedy knapsack algorithm that can make better decisions, but requires more accurate data.

To begin with, the authors consider a one-day variant of the problem. The authors state that a simple optimizer that uses less accurate models may outperform a more sophisticated optimizer that uses more accurate models. They claim that the importance of their paper is that they analyse the trade off between an optimizer's performance and model accuracy in the TAC/AA domain.

Building on the work of Tactex09[9], a TAC/AA prediction test was implemented that was used to evaluate the accuracy of the model used in the agent. The work carried out allows one to identify successful optimizers, as well as optimizers that would benefit from improvement. This builds on work done for the TAC

travel problem in the Greenwald et al.[20] paper. It can be concluded from this that the framework employed in both of these papers is effective and generally applicable.

Greenwald et al.[20] further state that their agent is distinct from others in that it reasons directly about bids that other agents will make rather than using the type of models developed by TacTex09[9]. This lends further credence to the idea that TacTex09[9] represents the state of the art in terms of modelling.

As discussed, Greenwald et al.[20] start by considering a more stylised version of the problem, which is much more tractable. Of the two proposed solutions, one is optimal under certain conditions and the other is a very good approximation. In this stylised version of the problem, the game is only one day in length and the daily capacity limit is a hard constraint. As such, the agent's goal is to choose per query bids that maximise profits without exceeding the budget. This problem is an instance of the non-linear knapsack problem. Although nearly optimal, the greedy knapsack algorithm stipulates fairly accurate estimations of the revenues, costs and sales functions.

The optimal solution that solves the knapsack problem has its drawbacks however. The greedy algorithm requires very accurate estimations of the revenues, costs and sales and so the paper's authors recommend the second approach – namely a rule based algorithm. This algorithm only requires us to estimate the conversion probability. Essentially this algorithm searches for a target return on investment.

This approach does not necessitate the need to differentiate the game functions and instead, only the estimation of the function is needed.

1. If the sales of the previous day exceed capacity; increase the targetROI (equivalent to decreasing the amount of sales).
2. If the sales of the previous day did not reach capacity decrease the targetROI (equivalent to increasing the amount of sales).

Once the ROI has been estimated then the corresponding CPC can be calculated and so a bid can be made.

However the TAC/AA game is more complicated, as there is a cross dependency between queries and different days. Greenwald et al.[20] first propose a solution to the cross-query dependency problem and then adapts it for the TAC/AA setting. The cross dependency is best modelled as a penalized MCKP (multiple-choice knapsack problem). Three methods are introduced for solving it:

1. Exhaustive Greedy PMCKP: A computationally expensive, but nearly optimal solution to the penalised multiple-choice knapsack problem. (PMCKP)

2. Dynamic Greedy PMCKP - Much more efficient than pure greedy PMCKP. It achieves this by changing its parameters at each iteration of the algorithm.
3. Hybrid Greedy PMCKP. This is a combination of the two approaches and boasts a complexity of $O(n^2)$ over the Dynamic algorithm's $O(n^2 \log(n))$.

Greenwald et al.[20] test the efficacy of the different algorithms they propose and compare them to their theoretical performance. In the artificial environment the hybrid MCKP algorithm performed best but when placed inside the TAC/AA environment the rule based algorithms performed best.

4.2 Improving the state of the art

TacTex09[9] and Greenwald et al.'s[20] agents performance can be seen as a watershed in the TAC/AA environment. The particle filter was used extensively in TacTex09[9].

An alternative approach could have to use a Viterbi algorithm.[22] This algorithm attempts to find a sequence of hidden states known as the Viterbi path that results in a sequence of observed events; in this case the observed events would be the detail from the previous day's reports and the hidden states the user populations, bids etc. The Baum-Welch algorithm is another possibility. It is a special case of expectation maximisation. A more fashionable alternative, and certainly one with most literature surrounding it would be to have taken a Bayesian approach. Bayes can tell us given the data, what is the probability that a certain vector of parameters generated it. Here, the Bayesian approach would have taken the form of Gibbs sampling. The Tactex09[9] bid estimation is mostly performed using statistical techniques.

An alternative approach would be to use machine learning models to build a picture of rival agents' behaviour and so to predict their bids from this. The most obvious improvement to all agents reviewed would be to include a way for the agent to draw on experience from previous games. This would improve the performance of certain machine learning agents (e.g. neural net based ones) as their training times can preclude them from a good performance early on in the game.

References

- [1] M. Wooldridge, *An Introduction to MultiAgent Systems - 2nd ed.*, Wiley, pages 293-302, 2009
- [2] S. Parsons, M.J. Wooldridge, *Game Theoretic and Decision Theoretic Agents*, 2000

- [3] J.S. Rosenschein, M.R. Genesereth, *Deals Among Rational Agents*, 9th International Conference on Artificial Intelligence, 1985
- [4] H.A. Simon, *A behavioral model of rational choice*, Quart. J. Econ., vol. 59, pp. 99–118, 1955
- [5] H.A. Simon, *Rational choice and the structure of the environment*, Psychological Review, vol. 63, no. 2, pp. 129–138, 1956
- [6] H.A. Simon, *Invariants of human behavior*, Annu. Rev. Psychol., vol. 41, pp. 1–19, 1990
- [7] W. Stirling, M. Goodrich, *Satisfying Equilibria: a Non-Classical Theory of Games and Decisions*, Proceedings of the First Workshop on Game Theoretic and Decision Theoretic Agents, pages 56–70, 1999
- [8] Y. Vorobeychik, *A Game Theoretic Bidding Agent for the Ad Auction Game*, http://vorobeychik.com/working_papers/qt.pdf
- [9] D. Pardoe, D. Chakraborty, P. Stone, *TacTex09: Champion of the First Trading Agent Competition on Ad Auctions*
- [10] D. Pardoe, P. Stone, *TacTex-05: A Champion Supply Chain Management Agent*, AAAI06, July 2006
- [11] D. Pardoe, P. Stone, *Designing Adaptive Trading Agents*, ACM SIGecom Exchanges, Vol. 10, No.2, June 2011
- [12] Y. Vorobeychik, *Simulation-based game theoretic analysis of keyword auctions with low-dimensional bidding strategies*, Twenty-Fifth Conference on Uncertainty in Artificial Intelligence, 2009
- [13] S.L. de Medeiros Rivero, B.H. Storb, R.S. Wazlawick, *Economic Theory, Anticipatory Systems and Artificial Adaptive Agents*
- [14] P. Stone, R.E. Schapire, J.A. Csirik, M.L. Littman, D. McAllester, *ATTac-2001: A Learning, Autonomous Bidding Agent*, <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.16.871>, 2002
- [15] E. de Graaf, *Trading Agents: Learning to Procure Goods and Services*, <http://www.liacs.nl/~edegraaf/Scriptie0228311.pdf>, MSc Thesis, June 2004
- [16] C. Eder, *Q-Learning*, <http://knol.google.com/k/q-learning>
- [17] Watkins, Dayan, *Q-Learning: Machine Learning*, 1992
- [18] M. He, N.R. Jennings, *SouthamptonTAC: Designing a successful trading agent*, <http://eprints.ecs.soton.ac.uk/6875/>, 15th European Conf. on AI (ECAI-2002)

- [19] M. Munsey, J. Veilleux, S. Bikkani, A. Teredesai, M. De Cock, *Born to Trade: a Genetically Evolved Keyword Bidder for Sponsored Search*
- [20] J. Berg, A. Greenwald, V. Naroditskiy, E. Sodomka, *A First Approach to Autonomous Bidding in Ad Auctions*
- [21] A. Greenwald, V. Naroditskiy, S. Lee, *Bidding Heuristics for Simultaneous Auctions: Lessons from TAC Travel*, Workshop on Trading Agent Design and Analysis, July 2008
- [22] G.D. Forney, *The Viterbi Algorithm: A Personal History*, 2005