

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.

5 Agent Implementation and Testing

Over the duration of the project we have read and evaluated a number of different papers, all of which provide different ideas as to how to go about implementing a successful agent for the TAC/AA simulation. In our literature review

we discussed many of these papers and highlighted the advantages of each.

In this report we will show how we used these papers to come up with some of our own ideas for agent bidding strategies. We will discuss which strategies we considered and which of these we felt would make successful agents. From there we will give details about some of the agents we implemented, how well they performed and why we chose our final agent. The report will round up with an evaluation on the performance of our chosen agent.

5.1 Agent Strategy Designs

In this section of the report we will discuss, in detail, some of the agent designs we decided to consider and what we thought of them.

5.1.1 Set Bidding

Our initial agent design was a very simple one, which had the aim of winning by just bidding more than the other agents and hoping to avoid making a loss. This first agent was designed in such a way that it would only bid set amounts for the different queries. How much was to be offered as a bid would depend on the query's focus level. Focus level 2 queries (F2) would take priority over those with F1 and F0 focus levels, as they were more likely to achieve profit [23].

The adverts that the agent would provide for F2 queries would be set to match the query exactly, as it would not be advantageous to do otherwise. However, with F1 queries, any 'null' values in the query would be replaced by the agent's special value for that field (manufacturer or component accordingly). This was to help increase the chances of selling items that would earn the agent more profit. On the other hand, a generic ad was provided for F0 queries. This design decision was made based on the results presented by Vorobeychik et al.[8], who implemented this as part of their strategy and claimed to get good results from it.

As this initial agent would not adapt its bids during the simulation, we decided that a strategy that made the agent bid a lot for queries would mean it would be very likely to win a majority of the bids. Therefore, we decided to bid between £10 and £50 for the different focus levels, with F2 queries using the agent's special manufacturer and component being given the highest bid value.

When implemented, this proved to be a very effective strategy for the most part, as the agent was very likely to win the bids. On top of this, due to how the auction system works, it would only have to pay a little bit more than the

second highest bidder. As a majority of the other agents in the simulation bid less than £2 on average, our agent made a lot of profit. However, we realised that this was a dangerous strategy as other agents could easily cause us large amounts of debt by also bidding high amounts, but still less than our bid (for example, bidding £40 when we were bidding £50). This would mean our agent would be forced to pay more than this high amount (i.e. £40.10), while the agent who bid it would only have to pay the next lowest amount (closer to the more normal bids of roughly £1-2). Being forced to then bid i.e. £40.10, would quickly cause our agent to lose all of the profit it had made and end up in large amounts of debt. We therefore decided that this agent would be inappropriate for the competition and so kept it purely for testing our other agents.

5.1.2 Adaptive Bidding

The idea of this second agent was to try and build upon some of the ideas presented in the first and introduce adaptive bidding. However, unlike the previous agent, this one started off bidding fairly low amounts in the range £0.10-£0.80, split across the different focus levels, and then increasing or decreasing them as required.

This agent focuses on the F2 queries, bidding only very small amounts for F0 and F1 queries. Additionally, for F2 queries matching the agent's speciality item and manufacturer, even more was bid. The agent makes use of the sales reports provided in order to determine the performance of its current bidding strategy. The agent monitors values for the past five days (the size of the distribution window for the simulation) for the following pieces of data:

1. Quantity being sold for each query. If the agent isn't selling many items for given type of query, q , then the agent increases how much it is willing to bid by a set amount. This amount varies depending on the focus level of the query and whether the query matches the agent's special manufacturer and component. The agent will increase the amount it spends if, on any of the previous five days, it has managed to sell less than 60% of the stock for that query is being sold. Similarly, if the amount of stock sold on one of those days goes above 90% then the agent will decrease the amount to be bid. Testing showed that taking this value, as opposed to finding the average amount sold provided better results, with more profit being made. The agent appeared to hover at selling 125% of the stock available, a value that Berg[24] claimed could provide behaviour closer to optimal behaviour.
2. Average position achieved for each query. The average position achieved over the five day period is monitored and, if the position drops below a chosen value, the bid for that query is increased, a strategy suggested by Kitts[23]. As with the quantity sold, the amount that the bid is increased

by varies depending on the focus level of the query and whether it matches the agent’s specialities. For F0 and F1 focus levels, the bid can also be decreased if it is felt the advert is doing too well. This design decision was made after reading Kitts[23], which claimed that an agent could do better overall if it wasnft always reaching the top advertising position as aiming for a spot lower could save a lot of money.

3. Finally, the agent will only consider bidding on a query based on whether the query has lead to more purchases than the advertiser can provide stock for. If too much stock has been sold, then the agent will cut all bids to a small value, in order to prevent continued over-selling, which can cripple an agent’s ability to make profit.

This agent appeared to perform very well in the initial testing that it was put through, and with some trial and error was able to make a fair amount of profit from an early stage in its development. We therefore decided to make this one of our main agents.

5.1.3 Evolving Agents Using Genetic Algorithms

Munsey[19] introduced the idea of using genetic algorithms in order to evolve an agent that could do well in the TAC/AA simulation. Using the ideas presented in this paper, we were also able to construct agents which were being evolved through use of a genetic algorithm.

As described in Munsey[19], the agents produced using this approach were modelled as finite state machines. Each of these finite state machines had seven states which the agent could currently be operating in, which was calculated using the formula:

$$S_d(x) = \begin{cases} s1, & \text{if } B_{d-1}(x) \leq -101 \\ s2, & \text{if } -100 \leq B_{d-1}(x) \leq -51 \\ s3, & \text{if } -50 \leq B_{d-1}(x) \leq -26 \\ s4, & \text{if } -25 \leq B_{d-1}(x) \leq -11 \\ s5, & \text{if } -10 \leq B_{d-1}(x) \leq 0 \\ s6, & \text{if } 1 \leq B_{d-1}(x) \leq 10 \\ s7, & \text{if } 11 \leq B_{d-1}(x) \end{cases}$$

where $B_{d-1}(x) = \sum_{i=d-W}^{d-2} c_i(x) - C(x)$. $B_{d-1}(x)$ defines the backorder, which is the amount of stock remaining to be sold from the past $W = 5$ days without receiving a penalty. A positive value indicates overselling, and negative underselling[19]. As the agent should at least try to ensure that all capacity is sold without going over the capacity threshold where possible, the ranges are not equal and become smaller as the backorder

decreases to zero, so that the bids for queries can be altered on a much finer scale. Going over capacity is not good, hence the single matching state for anything over a small positive backorder, which tries to reduce bid values as fast as possible to avoid the penalties associated with overselling.

In addition to this, we also implemented action tables for each state, which contained information for the agent on how to alter its bid from the previous day. For the initial population of agents, the action tables were randomly populated with values in the interval (-1,1), however, each successive generation used values inherited from its parents.

Munsey[19] also make use of the average position of a query to locate which row in the action table the agent should use to update the bid amount. The agent assigns a speciality match rating to the query based on how well it matches to what the agent is specialised in. These ratings can be found in the following table:

0	2 misses	VERY BAD	[19]
1	1 miss, 0 hits	BAD	
2	1 miss, 1 hit	WEAK	
3	0 hits, 0 misses	NEUTRAL	
4	1 hit, 0 misses	GOOD	
5	2 hits	PERFECT	

The integers in the first column represent the columns in the action table, so that, when used with the average position value, the agent can retrieve a value which is used to modify the previous day’s bid amount, in accordance with the formula:

$$b_x(q, d) = (1 + A_{i,j}^s) \cdot bx(q; d - 1)[19]$$

where $A_{i,j}^s$ is the action table for the current state of the agent, and i and j are the row and column values defined as the average position and speciality matching value.

The fitness function was defined, in accordance with Munsey[19], to be the average profit of the agent, after a minimum of three consecutive bidding games. This helped to ensure that an agent wasn’t chosen to become a parent for the next generation due to a random highly successful profit from one game. As became clearer later, after having performed numerous iterations of genetic algorithms, a possibly superior fitness function would have been the profit of the agent divided by the capacity of the agent, for that run. This would have taken into account the agent performing comparatively poorly against another agent when looking directly at profit, when the poorer performing agent had a much lower selling capacity.

Initially, a series of agents were produced with the action tables randomly populated with values randomly generated in the interval (-1,1). The action tables

were represented in Java using:

```
HashMap<Integer, double[ ][ ]>
```

which took an Integer, representing the state to return a 2D double array, of which the rows represented the average position of a query rounded to the nearest integer, and the columns represented the speciality matching.

One key aspect of most genetic algorithms is the need to introduce mutations to ensure that as more generations are evolved, they do not converge to a local maximum. In the case of the agents, this occurs when the values within the action tables become optimal from having starting from just one set of parents, whilst there was actually a set of values for the action tables which would result in a larger profit. We decided to not implement direct mutations produced in this manner within our agents. We felt that the use of BLX- α crossover to produce a range of values for the next generation of the agents to take introduced enough variation for mutation to not be required.

To perform the genetic algorithm, we applied the fitness function to all agents produced during a generation. From the results of this, we select the two fittest agents to proceed to the next stage of reproduction. As mentioned previously, we did not use the approach by Munsey[19] and instead randomly selected between the corresponding values in each agents action tables. This ensured that values from both parents were being directly passed on, which we believed would produce a better convergence to optimality over BLX- α crossover.

We kept the genetic algorithm separate from the agent itself so as not to lose track of what part was the algorithm and what part was the agent. The program produced to implement the genetic algorithm can be found in Appendix A. It reads in two comma delimited strings representing the values stored within the two best agents from the current generation. It then performs a random choice between the corresponding values from each string to produce a new offspring. Running this as many times as needed produced the population for the next generation.

Whilst Munsey[19] has proven that an agent developed using a genetic algorithm is feasible, and performs quite well, we felt that the time required to produce an agent which performed well in all scenarios was counter productive. This was also due to the better profits we observed from both of the other initial implemented bidding strategies. Whilst we understand that our initial genetic agents were in their infancy, the time taken to let them improve could instead be spent in improving the bidding strategies for the other two agents we produced, as they also produced a higher profit more consistently.

We also noted curiosities with some of the agents produced before we switched our approach to another bidding strategy. Sometimes an agent would be performing extremely well and be generating the most profit out of all other agents

competing in the same auction, but inexplicably start to make an ever increasing loss. Whilst this behaviour would hopefully eventually be removed from the agents being produced many generations into the future, it was also worrying behaviour.

5.1.4 Game Theoretical Strategies

We first considered how a game theoretic approach could be applied during the development of a TAC ad auction agent. Vorobeychik discusses the limitation of using game theory in a competitive agent setting:

”‘One reason that agent designers often eschew game theoretic techniques is that in general there may be many equilibria, and the problem of equilibrium selection requires coordination among the agents. Additionally, any asymmetric equilibrium requires coordination on roles. Finally, other agents may be imperfectly rational in a variety of ways (for example, buggy).’” [8]

However, by using the techniques discussed in the above paper to estimate value per click and adjusting bidding policy appropriately, the authors succeeded in programming a fairly successful agent.

The key points of this kind of agent are:

1. The estimation of the advertiser’s value per click
2. Finding an optimal value for the ‘shading factor’.

Regarding the point number two, the authors go into a quite detailed discussion about what values to assign to this shading factor, proving that against competent agents a value of 0.2 would be the dominant strategy and arguing that, against less efficient agents, an increased value would be preferable.

In order to achieve the first point however, the authors run more than 100 offline simulations, eventually having an estimations for parameters such as the distribution over the three different states of the user population for each of the 60 days duration, the sales per day and the click-through rate. Not having enough time to set up and run all those simulations, we decided that we could have used instead the techniques and strategies described by the TacTex09 paper[9] to estimate the user population, and the sales per day. Hence we decided to start with the implementation of the position analyser. This module of the TacTex2009 agent uses a search tree over a space of linear systems in order to give quite an accurate estimation of every other agent’s number of impressions. Unfortunately, we realised soon enough that this procedure relies quite heavily upon having the exact estimations for the opponents’ average position

and that, in the 2010 competition, only rough approximations are handed out by the publisher, based on a random sample on impressions. Not knowing how accurate estimations based on approximations would have been, we decided to abandon this path as well.

5.1.5 Cyclic agent

Being unable to predict what other agents will do in the actual competition (or, in other words, how efficient they will be), we considered the idea of an agent that would constantly adapt its bids in order to achieve certain positions in the impressions. For each query a lower and an upper boundary are hard coded into the agent, as well as a set total amount which represent an aggregated spend limit. The agent proceeds then to divide this spend limit between the different queries, giving different weight to each different query. A lot of effort has been put into finding the right balance between these different queries, especially when deciding whether to favour an F1 level query with component or manufacturer matching over an F2 query with no specialised component or manufacturer. In the end, we observed that most of the net earning would come from the selling of specialised merchandise, thus hinting us into considering only the F2 levels queries for items with at least either the component or the manufacturer matching the agent specialisation and favouring F2 queries with both of the items matching.

From observations of actual games, the better performing agents always kept the overall capacity usage between 125% and 150% of the total distribution capacity. This observation, however, is quite meaningless if not seen from the right perspective. For example an agent might reach a constant 150% capacity usage, only by bidding really high on all queries and not considering the "expenses", for these reasons we decided to impose some upper boundaries on the bidding. In order to determine what upper boundaries to apply, we modelled our strategy after the guidelines from [8].

We simplified the model even further, from empirical observations of the on-line simulations we saw that:

- F0 queries get converted with an average probability of 10%
- F1 queries get converted with an average probability of 18%
- F2 queries get converted with an average probability of 25%

These probabilities increase further on when the query is matching one or both of the agent's manufacturer and component specialities.

At this point we gave a very simple estimation of the real value of each query

for the agent by multiplying this focus-related probability by the revenue from a conversion. Following the prescriptions of Vorobeychik’s work[12][8], this value is further multiplied by the shading factor value, that we decided to set to 0.5.

Another feature that we decided to implement is the cyclic bidding mechanism, indirectly inspired by TacTex09[9]. That is, the daily spending limits follow a cyclic fashion, getting notably reduced every few days. There are two main reasons for this choice:

1. The estimation of each day’s sales is much simpler, because through this mechanism we can model the sales exactly the way we want. An extreme example is that of spike bidding: the spending limit is risen a lot every 5 days and the rest of the time is set to 0, when it is time to raise the spending limit we know exactly how many conversions we had the last time, and we know that the distribution capacity is completely free. Of course this is not advisable.
2. To ”‘unload’” the distribution capacity usage. This is a desirable event because, with the increase of the distribution capacity, the probability for a user to make a conversion decreases. If we had full information about the user population, the best time to unload the capacity window would be in those days in correspondence with the burst in the search behaviour, when it is more likely that users will search for an item without being in the focused state, and thus not making a conversion. Not having implemented any kind of predictions regarding the user population, however, such a mechanism will be likely to lower the spending limit in correspondence of at least some of these burst, beating for this reason an agent that instead would keep a lower and constant spending limit.

The last feature is an ”‘emergency’” mechanism, the need of such a mechanism has been clear when we took notice that the performance of the agent is partially influenced by the strategies implemented by the other agents and, most importantly, by how many competing agents have the same specialisations as ours in a given simulation. Each day, the agent computes what the average distribution usage is over all of the past days, if this mean is less than 125% of the distribution capacity, the agent will think that for some reasons it’s not making enough money, and it will sensibly increase the spending limit for the F1 queries that match either the component or the manufacturer specialisation.

Using these strategies, two different versions have been modelled and tested:

- The first one with a cycle of 5 days that alternates a spike, a break, and three days of levelled spending limit. In this version the days for the emergency check are set at the first and the last day of the three days of consecutive, levelled biddings.
- The second one with a cycle of 4 days that alternates a break and a 3 days levelled spending limit (which is set to a higher value than the first one).

In this version the emergency check is done every third day of the 3-days sequence.

5.1.6 Machine Learning Approaches

In the literature review, we saw that in the setting of the TAC ad auctions, machine learning could be a powerful approach. We therefore decided to investigate different forms of Machine Learning that may have been applicable to the TAC ad auctions setting.

We first considered using Artificial Neural Networks in order to 'train' an agent to the TAC/AA environment, a strategy mentioned by Bishop[25]. Once trained, this form of agent would have a quick response time, which would be vital due to the online nature of the TAC ad auction environment. Although the activity of such an agent may not be explicitly clear to the user, this is not a major disadvantage in this situation.

Unfortunately, this approach requires a significant time investment in order to reach a suitable level of competency. If left to train in a game, even if left to observe for a short time, it is unlikely the agent will be ready to act in the available time-frame; the 60 seconds over which a game takes place.

This concern could have been addressed by giving the agent a large data set to train on before it plays in a game, however, no such data is available to us and there is insufficient time and scope in the project to generate meaningful data.

Another machine learning strategy considered was Q-learning, as discussed by Watkins[17]. Consisting of simple algorithms, Q-learning is an implementation of Reinforcement Learning to find a locally optimal payoff.

This form of machine learning seemed like an attractive prospect; although many environments are too complex for this form of machine learning, aspects of the TAC ad auctions environment may prove simple enough to be modelled by this method.

This approach suffers when the set of possible states and actions becomes larger, making the space over which the algorithm must act intractable. This would decrease the efficiency of the algorithm and hence, the performance of the agent.

Papers discussed in the literature review demonstrate the potential of machine learning approaches to the problem of an TAC ad auction agent. Despite this, in the implementation, we decided not to focus on machine learning approaches due to the lack of a suitable environment and the restricted access to suitable training data. This was because other agents in the designated online server would not be suitable as a source of learning data, due to their incomplete and transient nature. The other alternative was to run a server to generate learn-

ing data for the agent from available completed agents. For example, previous competition entrants. We felt that this approach may lead to a biased data set and ultimately, inappropriate behaviour.

6 Strategy of Final Agent

As mentioned in the previous section of the report, we investigated into a number of different agents. We ruled out the use of agents based on machine learning and game theory and the agent that relied on a set, high bid for the reasons described. This left us with three other agents that could be possible solutions for the coursework. These were the adaptive bidding agent, the genetically evolved agent and the spike bidding agent.

In order to choose which of these agents to continue with we had to monitor how they each performed in simulations where they had to compete both against each other, and against other groups' agents. Each agent participated in a large number of simulations to ensure that they were tested at different distribution capacities and to help determine how consistent their results were. The table below shows the average profit achieved by each of the three agents when set at different distribution capacities.

Agent/Distribution Capacity	300	450	600
Adaptive Bidding	30k	45k	50-60k
Genetically Evolved	-5-15k	20-35k	35-60k
Cyclic Bidding	30-35k	45-55k	55-60k

This table was generated from data collected over a range of different simulations, where the agents involved were competing not only against each other, but also agents from other groups. As we had a number of different genetically evolved agents, we chose the one that performed best (it achieved the highest profits on average) for this table. This agent was one of the later generation agents.

As the table shows, the adaptive and Cyclic budding agents performed a lot better than the genetically evolved one, which was a lot more inconsistent with its results and performed a lot worse than the other two when on the lower distribution capacities. For this reason we decided to stop further development on the agent.

The table also demonstrates how similar the profit margins were between the adaptive and Cyclic bidding agents. These two agents both performed very well in nearly all of the simulations they were tested in. When at the same distribution capacity, both would finish the simulation with very similar profit margins. When competing against other agents (dummy agents, or those from

other teams) the agents continued to perform nicely and would often place first and second dependent on competitors' distribution capacities.

In the end we decided to turn the Cyclic bidding agent into our final agent and used the other to continue testing our chosen agent. This decision was made because the Cyclic bidding agent performed ever so slightly better than the adaptive bidding one. This was likely due to the reasons discussed in the Cyclic Agent section, that is, the small likelihood of a spend limit decrease in correspondence of an interest burst in the user population, while, in contrast, the adaptive bidding agent had no ways to avoid these spikes in the Informational State user population, having always a set spending limit.

Between the two different versions of this agent, we chose to use the second one, as it showed to be more reliable in delivering good performances under all situations, including the cases where more than one of the competitors had the same specialisations as the agent tested.

6.1 Report on the Cyclic Agent

This agent can be seen as the cooperation between three different modules:

- the true value estimator
- the bidding estimator
- the spending limit estimator

6.1.1 The True Value Estimator

This part of the agent is what, at the beginning of each simulation, establish what is the upper bound in the value of the bid for each one of the query we're interested in as discussed in the section Cyclic Agent.

This value is obtained by multiplying three factors: the profit from the sale of the product identified by the query, the percentage related to different focus levels and the shading factor. This should model the function where, for each query q , the value that the query has for the advertiser:

$$V(q) = Pr(\text{conversion}|\text{click})E(\text{revenue}|\text{conversion})$$

as suggested by Vorobeychik et al.[8].

6.1.2 The Bidding Estimator

This is the part of the agent in charge of, once received the desired upper and lower bounds, calculating what the required bid is in order to place the advertiser's ad in a position between the given boundaries, constantly adjusting this value as the simulation progresses.

To achieve this, the agent keeps in memory, for each query and for each day of the simulation, the average position achieved and the corresponding bid. All of these bids are then used in the computation of a weighted mean making a distinction between four different cases:

- An average position strictly above the lower bound and below the higher bound, in this case the bid will have a full weight in the mean computation multiplied by a small factor (1.05). This slightly increased weight is placed there as a possible counter measure to other agents trying to increase their bids as well.
- An average position below the lower bound, in this case, before adding the value of the bid divided by the number of days to our mean, we multiply this value by an amount proportionally increased with the distance of the average position from the lower bound.
- An average position above the upper bound, this means that the bid was too high, and we need to reduce it in order to achieve the desired position. This is done by multiplying the bid value by an amount proportionally decreased with the distance between the average position and the upper bound.
- A NaN average position, this means that the agent didn't have any impressions for that day, in this case the bid gets increased by a fixed factor that gets decreased by an amount proportional to the value of the upper bound.

At the end of the whole computation, the module will output the minimum between the calculated mean, and the upper bound to the query bid.

This whole mechanism is absolutely useless when the desired position is the first one, that is because the TAC/AA competition is a second-price kind of auction, meaning that bidding the upper bound related to the query is the dominant strategy in order to get ranked in the first place.

Due to the nature of the game, for some queries and in some game situations, it might be desirable to spend a bit less and get in some position below the first one, because users will still have a chance to click on the subsequent adverts, depending on the continuation probability, which is randomly drawn by the server at the beginning of each game simulation.

6.1.3 The Spending Limit Estimator

This is the part of the agent in charge of establishing the daily spending limits. An internal clock, modulo the number of days in the cycle, dictates what is the action that the agent should undertake. In this 4-days cycle there are 3 different possible actions:

- When the clock is on 0, sets a low spending limit for F2 queries and a value next to 0 for every other query.
- When the clock is on a value greater than 0, sets a high spending limit for F2 queries that match the agent’s specialisations (even higher in the case of both manufacturer and component specialisation match) and a value next to 0 for every other query.
- If at any given day past day 2, the average distribution usage is less than 125% of the total distribution capacity, sets the spending limit for F1 level, specialisation matching queries to a medium value. This action is what implements the ”‘emergency’” mechanism discussed in the Cyclic Agent section.

7 Testing of Final Agent

This section of the report is designed to give details on how we tested our final agent. However, the strategies mentioned below also applied to testing the adaptive and genetically evolved agents as well and were useful in helping decide which of the three would become our final agent.

Each of the tests was run a number of times to try and determine how consistent the results were. If tests indicated a change needed to be made, then they were repeated.

The different tests we carried out are explained below:

1. Set bidders. This test involved testing our agent against two others that were programmed to only bid a set amount for each query. This amount would not change during the course of the simulation. The agent was tested against an agent who both bid quite high and another that bid low amounts.
2. Adaptive bidders. This involved testing the agent against an agent whose bid amounts would vary between days. The agent was programmed in a very similar way to our second agent.

3. Randomly generated agents. These tests consisted of adding some of the genetically evolved agents to the simulation with our agent. The competing agents included those from the earlier generations, as well as a few from the later, more evolved and more optimal agents.
4. Spike bidding. The fourth test was against an agent that would lay fairly dormant for a few days, coming across as being quite passive, before bidding a lot on a number of queries for a day and returning to a more passive mode. This test was added to see how well our agent could adapt to sudden changes in other agents' behaviours.
5. Distribution capacities. The final and most important test was to see how well our agent performed when assigned different distribution capacities. In many cases we found that an agent that performed well in other tests on a capacity of 600 would do quite poorly with only 300 or 450.

The results from these tests allowed us to keep trying to update our agent to provide closer to optimal bidding strategies. Overall we found that the cyclic bidding agent did well in each of the tests. However, it took a lot of trial and error in order to optimise the agent so that its bid to profit ratio was favourable.

In both the first and the second tests, the cyclic bidder performed well and managed to collect more profit than the other two agents. Our runner up agent, the adaptive bidder, performed marginally better than the competitor due to more optimised parameters and achieved similar profits to the genetically evolved agent.

The third test was one that our final and runner up agents were all able to perform well on. Although the genetically evolved agent we had considered using acted inconsistently, it was still able to achieve profits ranging from 30-50k (depending on the distribution capacity it was given). The other two agents performed equally as well, if not better.

When competing against another spike bidder, our final agent was seen to operate less successfully than in the other tests. However, it still did very well and managed to beat the competitor. Unfortunately, in this test, the genetically evolved and adaptive bidders did not perform so well and often achieved profits £5,000 smaller than our final agent. It was in these tests that we knew for certain that our spike bidder was our best agent.

Finally, each of our agents was tested in simulations in which they had the same or different distribution capacities. We ensured that each agent was tested on each possible capacity value in order to be sure that the agents could perform well regardless of how much stock they had available to sell.

Although all of the tests did go well, they nevertheless sometimes found problems with our agent and a lot of trial and error was required in order to optimise

the agent as much as possible so that it bid a sensible amount but still did well.

7.1 Cyclic Agent

It was found that this agent performed quite well on the majority of these tests, usually outperforming the other agents that were assigned the same distribution capacity.

This agent was able to adapt to other agents' biddings as long as those agents were not bidding irrationally high amounts, greater than the cyclic agent's value for those queries. In these cases the agent would just settle down for a lower position in the impressions, but without having to pay more than his actual value for the query. As it turned out, its main weakness is represented by sharing the same component and manufacturer specialities with competing agents, as it is not able to detect this type of situation and therefore it will not change its own strategy.

Agent	Capacity	Profit
micman	450.0	34516,66
manmic47	450.0	37299,07
manmic46	450.0	28540,03
manmic	600.0	34705,41
manmic45	450.0	27479,65
manmic49	600.0	39469,32
manmic48	300.0	20060,27
manmic50	300.0	25933,50

Fig. 1: This table shows the result of a test where micman is our Cyclic agent, and all the other agents are instances of the same Adaptive Bidding agent. Although the performance itself is not impressive (A final profit of 35k with a 450 capacity), it is to be noted that the agent was sharing the same manufacturer AND component specialities with three other agents. manmic47, being the only one with that particular manufacturer/component combo, outperformed our Cyclic agent. On the other hand manmic was one of the agents (along with manmic45 and manmic58) sharing the same specialities, and, with a capacity of 600 against the 450 of the cyclic agent, managed only to have a final profit of 200 bigger.

Agent	Capacity	Profit
micman	600.0	58005,78
manmic49	450.0	39768,07
manmic48	450.0	42993,33
manmic46	450.0	38352,45
manmic	600.0	51288,97
manmic45	300.0	21912,23
manmic50	450.0	31109,52
manmic47	300.0	21077,57

Fig. 2: This table shows the results of a test with the same agents from figure 1 being tested, with the only difference that in this case our Cyclic agent was not sharing the component/manufacturer specialities combo with any of the competing agents, it is obvious from this test how well it can perform in this kind of situations.

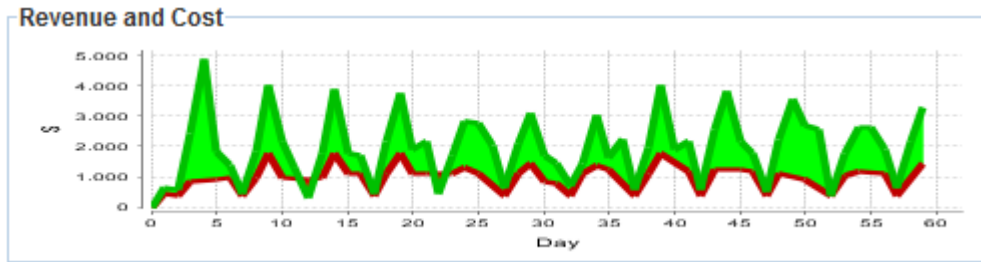


Fig. 3: This image shows the graph for revenue and cost from the same simulation as in Figure 2. From this picture we can fully appreciate the cyclicity in the spending limits of our agent.

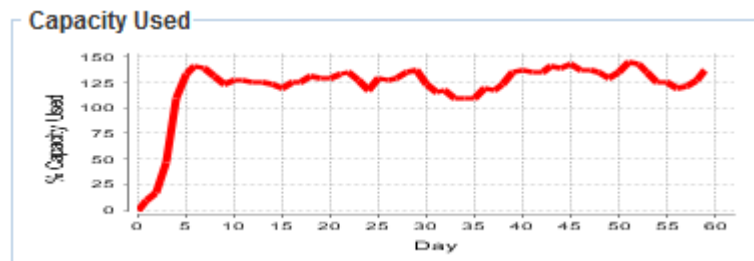


Fig. 4: This figure, taken from the same simulation as for fig.2 and fig.3, shows the capacity usage in percentage. We can appreciate here how the capacity swings between 125% and 150%, which are our target capacities.

This lead us to set up another test, featuring the two different versions of the Cyclic agent put up against six copies of the same adaptive bidding agent. The test was ran different times, every time with different distribution capacities and specialisations, and every time the Cyclic agent performed better than the opponents with the same capacity and specialisations. A few times a Cyclic agent

with 450 distribution capacity produced almost the same profit as an Adaptive Bidding agent with 600 capacity and the same manufacturer and component specialisations (as in the pictures above).

At this point the choice seemed obvious and another test was set up, featuring the second version of the Cyclic agent against 7 copies of the first version. In this tests, often the second version outperformed the first one, eventually leading us to the choice of the second version of this agent as our final agent.

7.2 Possible Improvements

Although our Cyclic agent performed decently well in most situations, there is still much room for improvement. It is clear at this point that its static strategy cannot adapt to situations where similar agents shared the same specialisations, severely undermining the agent performance. This is because they all bid for the same queries, causing the click price to raise until it reaches the upper boundary. If the agent could recognise these situations, it could also decide to spread the bids over different queries, maybe less valuable, but cheaper for the agent.

Another possible improvement would be to implement a user population estimator, that could help giving a dynamic estimation of the true bid value for the agent. The creators of TacTex09 used a particle filter to efficiently implementing this mechanism, unluckily we found ourselves lacking time for such a complex task.

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

- [23] B. Kitts, B. Leblanc, *Optimal Bidding on Keyword Auctions*, 2004
- [24] J. Berg, A. Greenwald, V. Naroditskiy, E. Sodomka, *A Knapsack-Based Approach to Bidding in Ad Auctions*, *Frontiers in Artificial Intelligence and Applications*, Volume 215, 2010
- [25] C. Bishop, *Pattern Recognition and Machine Learning*, Springer, 2006

A Appendix: Genetic Algorithm Program

```

package readinfile;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;

public class Main {

    public static void main(String [] args) throws
        FileNotFoundException, IOException {
        String fileName = "Agent1.txt";
        ArrayList<String []> population = new ArrayList<
            String []> ();
        ArrayList<Double> fitnesses = new ArrayList<Double
            > ();

        BufferedReader reader = new BufferedReader(new
            FileReader(fileName));

        //Read in 2 tables worth of data – needs to be
            generalised
        String [] table = reader.readLine().split(",");
        String [] table2 = reader.readLine().split(",");

        population.add(table);
        population.add(table2);

        //Calculate each table's fitness value
        for (int i = 0; i < population.size(); i++) {
            fitnesses.add(getFitness(population.get(i)));
        }
    }

```

```

//Select 2 parents
ArrayList<String []> selectedParents = new
    ArrayList<String []> ();
selectedParents.add(population.get(0));
selectedParents.add(population.get(1));

try {
    // Create output file
    FileWriter fstream = new FileWriter("out.txt",
        true);
    BufferedWriter out = new BufferedWriter(
        fstream);

    //Crossover genes
    Double [] child = crossover(selectedParents.get
        (0), selectedParents.get(1));
    for (int i = 0; i < child.length; i++) {
        out.write(child[i]+"");
    }

    out.write("\n");
    out.close();
} catch (Exception e) { //Catch exception if any
    System.err.println("Error: " + e.getMessage());
    ;
}

}

static double getFitness(String [] table) {
    //Find fitness value associated with individual (
    calculated as being the bank status at the end
    of the game
    String f = table[table.length - 1];
    double fitness = Double.parseDouble(f);
    return fitness;
}

static ArrayList<String []> selection(ArrayList<String
[]> currentPopulation, ArrayList<Double> fit) {
    int numberOfParents = 2;
    ArrayList<String []> parents = new ArrayList<String
[]> ();

```

```

        ArrayList<Double> rouletteSelection = new
            ArrayList<Double>();

        double selectionProb = 0;
        double totalFitness = 0;
        for (int i = 0; i < fit.size(); i++) {
            totalFitness += fit.get(i);
        }

        //Add cumulative probabilities to 'selection'
        for (Double f : fit) {
            double prob = f / totalFitness;
            selectionProb += prob;
            rouletteSelection.add(selectionProb);
        }

        //Select new parents
        for (int j = 0; j < numberOfParents; j++) {
            double rouletteSpin = Math.random();
            for (int k = 0; k < rouletteSelection.size();
                k++) {
                if (rouletteSelection.get(k) >=
                    rouletteSpin) {
                    parents.add(currentPopulation.get(k));
                    break;
                }
            }
        }

        return parents;
    }

    static Double[][] crossover(String[] table1, String[]
        table2) {
        Double[] child = new Double[table1.length - 1];
        // -1 to cut off the profit value at the end
        for (int i = 0; i < table1.length - 1; i++) {
            double val1 = Double.parseDouble(table1[i]);
            double val2 = Double.parseDouble(table2[i]);

            double newValLower = Math.min(val1, val2) - (
                Math.max(val1, val2) - Math.min(val1, val2)
            ) * 0.25;
            double newValUpper = Math.max(val1, val2) + (
                Math.max(val1, val2) - Math.min(val1, val2)
            ) * 0.25;

```

```

//Choose value between the allowed range (with
    uniform distribution)
double newval = newValLower + (Math.random() *
    (newValUpper - newValLower));
double scale = (newValUpper - newValLower) /
    2;

// Rescale between [-1,1]
newval = (-1) + (newval - newValLower) / scale
    ;
System.out.println("newval = " + newval + "
    newValLower = " + newValLower + "
    newValUpper = " + newValUpper);*/
double f = Math.random();

    if (f < 0.5) {
        child[i] = val1;
    } else {
        child[i] = val2;
    }
}

return child;
}
}

```