Trading Agent Final Report:

Tiger Blood

Melissa Cox, Augie Hill, Max Kolasinski

SI 652 / EECS 547

Introduction

Our approach to creating a successful TAC/AA agent primarily focused on simplicity of design and an incremental development process. By acknowledging the fact that we would not have enough time to successfully develop an intricate agent, we were able to discover a very simple yet effective algorithm that accounted for many complex components of the TAC/AA scenario. Starting with this simple algorithm, our incremental development process and custom heads-up display allowed us to empirically gauge the effectiveness of additional features.

Sources Consulted

We consulted the existing TAC/AA literature and focused on learning what made past winners such as TacTex, Schlemazl and QuakTAC [1] so successful. We additionally consulted the publication describing the lessons learned from the first TAC/AA tournament [2], from which we learned that the most important aspect to creating a successful agent is to manage distribution capacity well. We referred to the competition specifications [3] frequently so we understood the different parameters and how to exploit them. Additionally, we utilized custom graphical charts, which were automatically archived, beyond those made available on the server, so we could watch various metrics of our agent rise and fall as the game progressed and then later go back to the graphs and compare them with the current iteration. This turned out to be key in quickly analyzing what worked and what needed to be changed.

Process

We first reviewed the specifications of the competition as well as the literature describing past successful agents. We were drawn to the TacTex, Schlemazl and QuakTAC agents and tried to emulate their behavior in our own agent. After sketching out the factors we could use from the specification, we broke down the information we knew from the game initialization, what would be revealed to us about our own agent throughout the game and what we could learn about our competitors. We identified the following list of factors to consider:

- user model/ NSInit: predicting how many users at each state (NS, IS, F0, F1, F2, T)
- advertiser model: predicting number of impressions, query type and estimated bids of the competing agents
- ad type: keep track of all ads and assume the most frequent is the predicted type (generic or targeted)
- capacity: awareness of capacity, noting which level (low, medium or high) and predicting when our agent is near capacity.
- bidding: when capacity is high, bid aggressively; as capacity becomes lower, bid more conservatively
- specialties: bid aggressively on our own agent's F2 specialty
- spending limits: assume spending limits are same as previous day

We had a plan. However, when it was time to start developing the agent, we hit a wall. It seemed like we had bitten off more than we could handle. A change of direction was needed.

Instead of starting with a complex agent that would take a long time to code and would be difficult to debug, we turned that process upside down and started with the simplest bidding algorithm we could think of that might work: a hill-climbing algorithm.

Despite its simplicity, this agent performed well. At this time, we felt this was an appropriate base on which to begin integrating more sophisticated domain knowledge. It was also at this time that we began analyzing our agent with the help of graphical charts generated using the game data.

We understood from the readings that controlling capacity would be crucial to building a successful agent. There were many factors to consider, especially regarding the other advertisers, so for simplicity we decided to first start by optimizing capacity using a few simple heuristics. Iterations V1 and V2 focused on controlling capacity, and iteration V3 was a small change to the bidding algorithm.

After we did surprisingly well in the practice tournament using V3, we were feeling good about our simple agent, so decided to only make measured improvements to that algorithm for the final tournament. It was a good thing we tested new features against the last proven agent, because if we hadn't, we might have used V5 in the final tournament, which we found in testing would have been a disaster, and instead used V4.

Iterative Development

In order to gauge the effectiveness of our incremental changes, we ran our agents through practice games set up on a remote virtual Linux server. We iterated through 5 major versions of our agent before the final tournament.

Naive

The naive agent was a very simple profit hill-climbing agent that was intended to give us a baseline against which we could test our ideas. The pseudocode for each query of this agent is as follows:

INIT:

```
// set a starting price
if (queryType is F0) {
      bidPrice = USP * 0.04
} else if (queryType is F1) {
      bidPrice = USP * 0.06
} else if (queryType is F2) {
      bidPrice = USP * 0.1
}
// set an increment
bidDelta = 0.05 * bidPrice
```

UPDATE:

```
// update direction
if ((negative profit and positive bidDelta)
      or (profit from 2 days ago > profit from yesterday)) {
      bidDelta *= -1;
}
// update bid price
bidPrice += bidDelta
```

The INIT section sets the starting bidPrice and bidDelta for the query. The UPDATE section changes bidDelta according to the profit history and then updates the bidPrice. We tuned the INIT coefficients by playing against the Dummies, and eventually started winning most of those games.

V1

For iteration V1, after adding targeting for our F2 specialization, we focused primarily on controlling our capacity. We did this first by setting spending limits to 1 for a portion of the queries with the highest cost per conversion. The number of queries to cut off was linearly increasing proportional to the amount of capacity in excess of 50%. For example, if our capacity was at 40%, no queries would be cut. If our capacity was at 51%, 1 query would be cut. If our capacity was at 100% or above, all of the queries would be cut.
This ended up being slightly better on average than the Naive agent, but produced obvious problems when we looked at the capacity graphs (see Appendix A).

V2

For this iteration, we attempted to improve upon V1's capacity management. It was plain to see from the capacity graphs that the approach we took in V1 resulted in wild swings of capacity. We wanted instead to reach our capacity and stay at around that capacity. We realized by looking at our graphs and the relative conversion baselines for each focus group that the F0 and F1 queries were of little benefit once we reached our capacity, and that we would instead want to focus our effort on gathering conversions from just the F2 queries. So instead of cutting off the F2 queries, we would set a spending limit for each, excluding the manufacturer and component specialization query, that is (Avg Cost / % Used Capacity). This is a simple scaling of the expected total cost for the query on the next day by how far over capacity we are, and it worked well (see Appendix A).

V3

With our capacity problems solved, the pressing task was next to fix the behavior of the bidding algorithm. As shown in Appendix B, the algorithm would continue descending to 0 regardless of the minimum price to get an ad placed. This wasted time that could be spent in search of the right bid. The UPDATE method from earlier was changed to the following, with the new parts in red.

UPDATE:

```
// update the min
if (did not get an ad slot) {
      min = bidPrice;
}
// update direction
if ((negative profit and positive bidDelta)
      or (bidPrice <= min and negative bidDelta)
      or (profit from 2 days ago > profit from yesterday)) {
      bidDelta *= -1;
}
// update bid price
bidPrice += bidDelta
```

This was the agent we used in the practice tournament. It was simple, but effective.

V4

For this iteration, we made a few simple changes that were obvious after the practice tournament. The first was to target on all F2 queries because there is no risk of targeting incorrectly. The second was to only bid on the F0 query when capacity is HIGH. This was because we noticed we were wasting a lot of our limited capacity space on the low profit-per-conversion F0 query. When capacity is high, however, we found that gaining that small profit-per-conversion from F0 was worthwhile at the beginning of the game. The F0 query would always get cut off by the spending limit algorithm once we got nearer to capacity. The final change to this agent was to modify the start bid coefficients slightly. This was the agent we ended up using for the tournament.

Reflections on the Tournament

Using the V4 version of the TigerBlood agent, we placed first in the competition. We performed very well at medium capacity, which is fortunate since probability distribution dictates medium capacity is the most frequent. Our average was approximately $50,000 per game over 20 games. We were cable to capitalize on Game 2 with a high capacity and strong result of over $72,000. This may have been attributed to the fact that not all agents were playing, as we were never able to replicate this high score. We clearly under-performed at low capacity, but overall did very well for the tournament. BidBuddy was our closest competitor in terms of game average; when comparing overall game averages the difference was only ~$95. (Not counting zero games or partial games, just full games).

Conclusion

In conclusion, TigerBlood owes it's winning strategy to the iterative design process that quickly identified how important controlling capacity was. By repeatedly testing out new theories and reviewing the automatically generated graphs, we could fine-tune the hill-climbing algorithm and correct some of our mistakes (such as setting spending limits on F2 queries).

Why does the simple profit-seeking hill-climbing algorithm work so well in such a complex environment? We believe it is because the profit metric encapsulates much of that environment

into a single metric much the way the price of a good efficiently encapsulates the means of production and its relative value to society. By optimizing on that one metric, we are therefore implicitly optimizing on all everything that produced that metric.

Additionally, we were able to win the Final Tournament due to the fact that we dominated in early games when not all agents were playing. When comparing full games played, we beat BidBuddy by less than a hundred dollars, so we recognize that improving our performance during low capacity games would be crucial to competing in future tournaments.

References

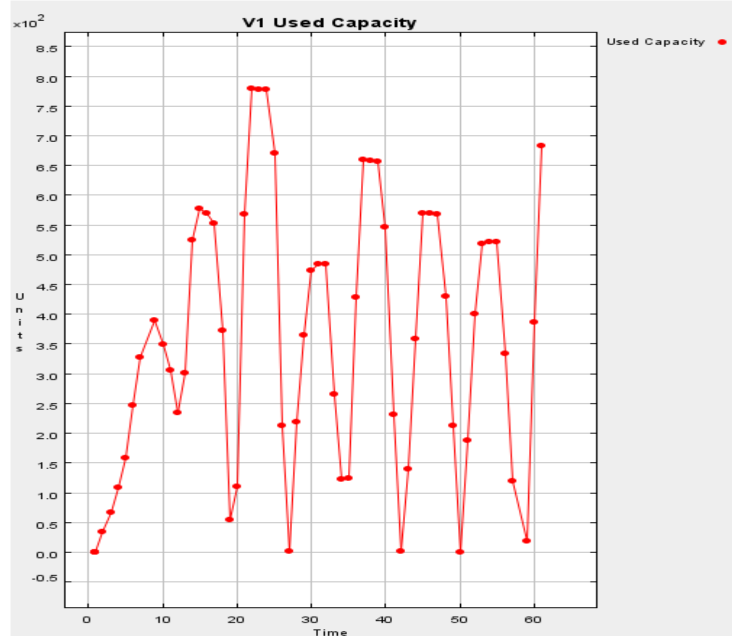
[1] Pardoe, D; Chakraborty, D; Stone, P. TacTex09: Champion of the First Trading Agent Competition on Ad Auctions. Technical report, University of Texas at Austin, 2009.

[2] Jordan, P R; Wellman, M.P.; Balakrishnan, G. Strategy and Mechanism Lessons from the First Ad Auctions Trading Agent Competition. Technical report, 2010.

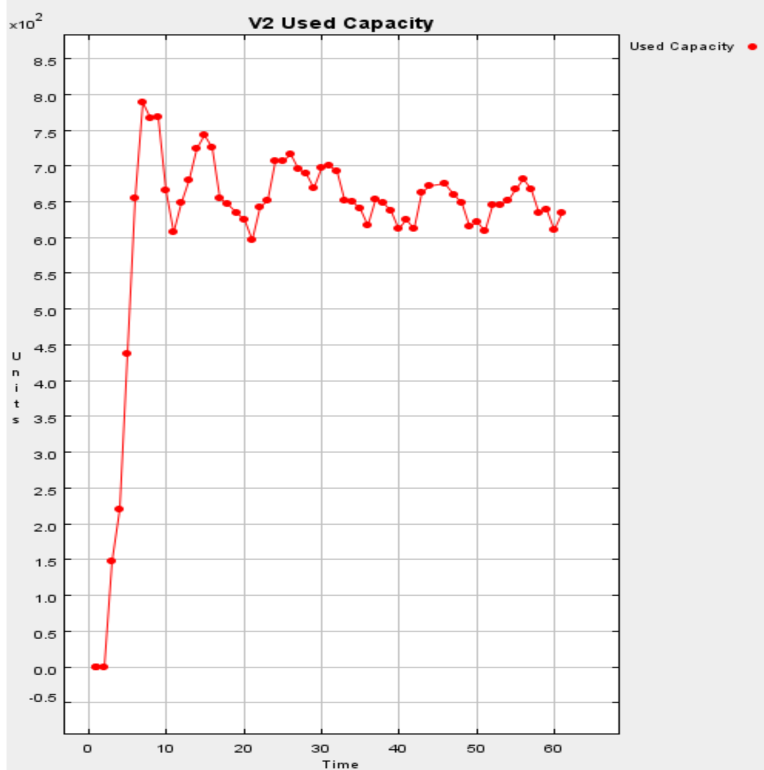
[3] Jordan, P.R; Kaul, A; Wellman, M.P. The ad auctions game for the 2010 trading agent competition. Technical report, University of Michigan, 2010.

Appendix A: Visualizing Capacity

This chart shows our V1 agent (capacity 450) and our first attempt at controlling capacity. When the agent would climb to near capacity, it set the bid limits of everything to 1. These dramatic spikes made it easy to see we needed to aim for a smoother capacity control.

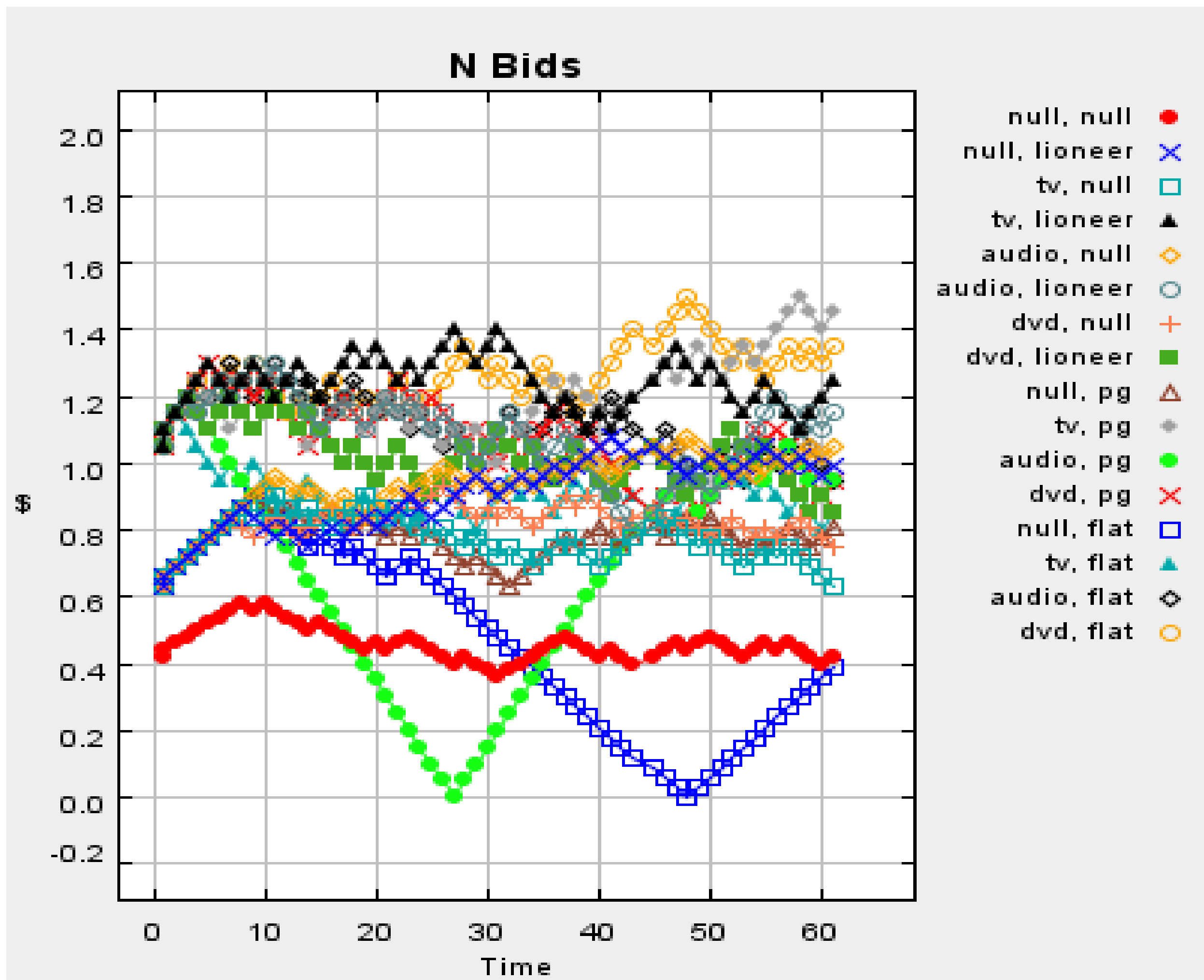


By taking a measured approach to setting spend limits on F2 queries, we could now more efficiently hover near or at capacity (here at 600). Although it went over capacity at times, the spikes are not nearly as dramatic as we saw in V1 (above).



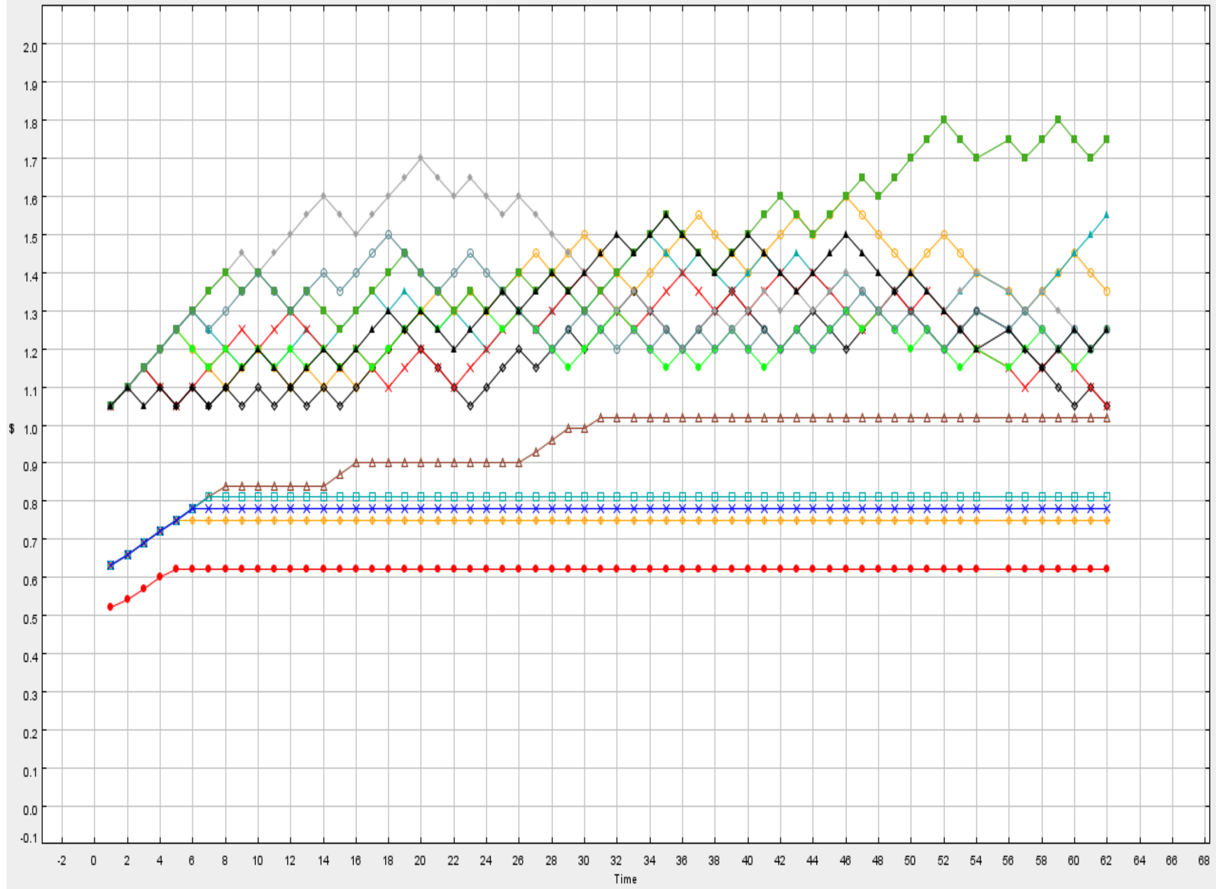
Appendix B: Bidding Algorithm Without a Floor

TigerBlood agents N to V2 had this odd behavior in some queries where bids would go all the way down to 0. V3 fixes this problem.



Appendix C: Cost Recovery on Under-performing Queries

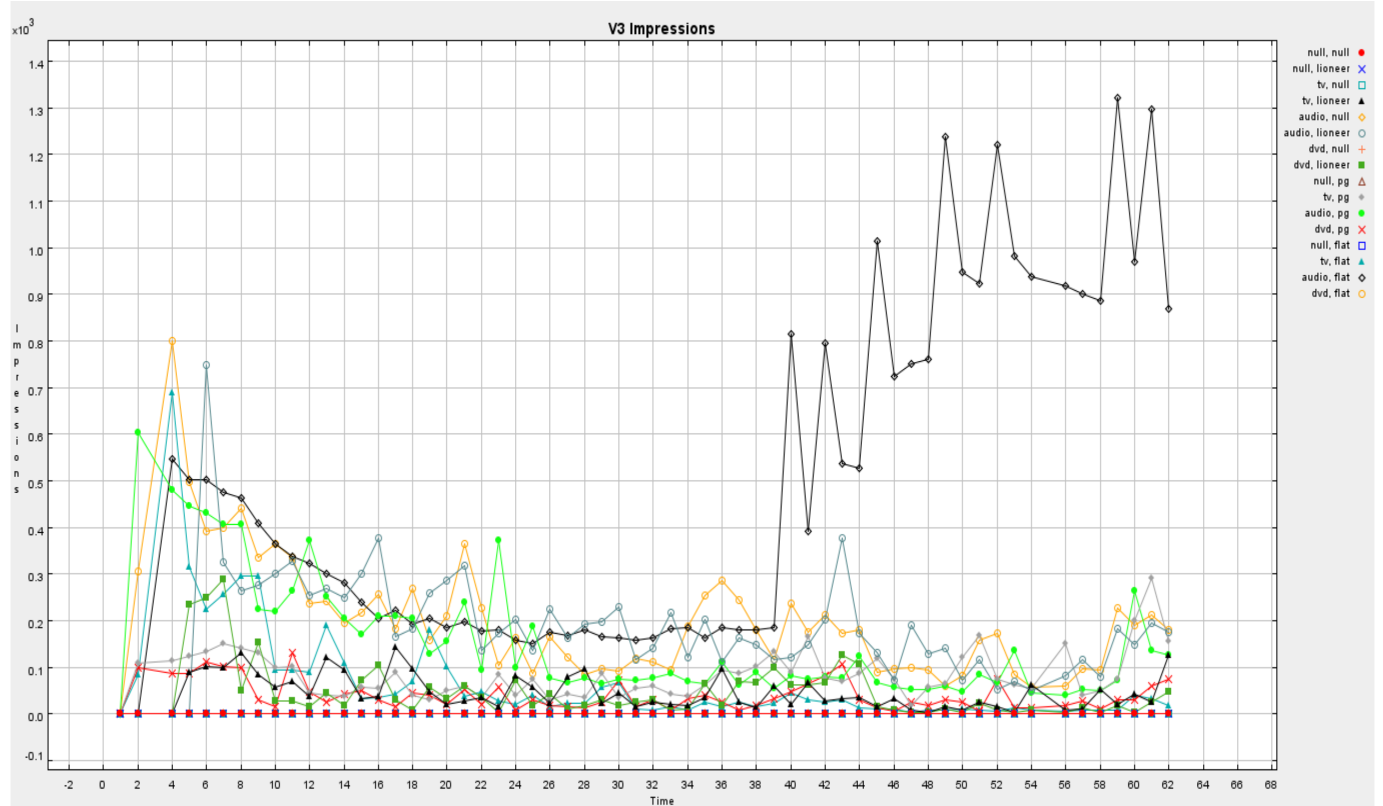
This graph illustrates our agent cutting off under-performing queries as the game progresses. In round 5, the [null,null] query gets dropped so we don't waste money on it. After this behavior was observed in the practice tournament, we decided to drop [null, null] entirely instead of wasting any money in the first few rounds- we know it gets dropped early and does not seem to help.



V3 Bids
$
Time
null, null
null, lioneer
tv, null
tv, lioneer
audio, null
audio, lioneer
dvd, null
dvd, lioneer
null, pg
tv, pg
audio, pg
dvd, pg
null, flat
tv, flat
audio, flat
dvd, flat

Appendix D: Visualizing Impressions and Estimating Specialties

This graph illustrates the number of impressions by query type. In this game, it is easy to see that our specialty is [audio, flat] which dominates for the first half of the game, then really takes off at round 39.



Appendix E: Tournament Results

Game	Rank	Cap	Result	What we observed
356	1st	450	53 236.37	• won at medium capacity • not all agents playing
357	1st	600	72 736.87	• personal best so far (highest result of our test results)
358	5th	300	33 283.28	• not doing well for low capacity • winners at high capacity not performing as well as we did when we had high capacity
359	3rd	450	56 216.53	• did very well for medium capacity • very close to second place (assuming 1st & 2nd slots had high capacity)
360	2nd	450	50 543.02	• second place for medium capacity • unsure if non-playing agents are assigned high capacity
361	1st	600	49 380.60	• 1st place at high capacity but should have done better than 49K!
362	5th	300	38 613.45	• good ROI, just not placing well due to lowest capacity
363	3rd	450	47 476.71	• overall, leading the tournament by ~21K
364	3rd	450	47 903.95	• EasyCommerce catching up! • assuming top 2 are high capacity • BidBuddy beat our high score (at 74K)
365	2nd	600	53 876.78	• close to dis_inc (winner) • didn't do as well as we'd like with high capacity
366	8th	300	25 091.81	• first instance of last place! • msmbot- assuming this is their first game at high capacity where they were playing • still winning overall, but margin is shrinking
367	3rd	450	45 328.38	• EasyCommerce 7th (low capacity) that will help
368	1st	450	54 064.03	• first place (by ~10K) for medium capacity
369	4th	300	44 011.21	• personal best for low capacity (42- impressive!)
370	1st	600	60 559.60	• realizing both our agent and EasyCommerce do really well with high capacity
371	2nd	450	53 171.05	• found our niche
372	1st	600	62 626.36	• leading by 11K+
373	2nd	450	47 026.67	• doing fine
374	1st	450	56 555.69	• first place, beating high capacity

				• beat BidBuddy by $27!
375	7th	300	33 117.49	• not our worst game

Appendix F: Final Tournament Results

This graph depicts the final tournament results, where we averaged 49,249 per game. Game 366 is our worst, where we scored just $25K with low capacity. Compared to other teams, we did not recover from low capacity games as well as other agents, but did very well for medium capacity.