Intelligent Agents: Agent Elman

Mohammad Ali Khan
University of Southampton
Montefiore House, Wessex Lane
Swaythling, Southampton
+44 74 12360633
mak1g11@soton.ac.uk

Fabrice Clarke
University of Southampton
131 Broadlands Road
Southampton
+44 74 12360633
fc1g11@soton.ac.uk

David Brewis
University of Southampton
131 Broadlands Road
Southampton
+44 74 12360633
db8g11@soton.ac.uk

ABSTRACT

This paper discuss the TAC Trading Competitions, mentioning different challenges faced. It then details the design and strategies of Agent Elman, mentioning the reasons behind different decisions. Lastly, it analyses the result of the competition and the agent's achievement of seventh position.

Categories and Subject Descriptors

D.3.3 [**Intelligent Agents**]: TAC Trading Agent Competition – *intelligent agents, trading competitions.*

General Terms

Algorithms, Management, Measurement, Documentation, Performance, Design, Economics, Reliability, Experimentation,

Keywords

Agent, trading, competition

1. INTRODUCTION

TAC (Trading Agent Competition) is a well-known competition used as a competitive benchmark for intelligent trading agents. It consists of a number of clients who need travel packages for a vacation over a notional 5-day period. They need to travel to the venue, stay in a hotel and possibly attend entertainment events during their stay. Each game consists of 8 agents and lasts 9 minutes. In these games, agents have to buy flights, hotel rooms, and trade tickets to complete the best packages for their clients, receiving most utility for the least cost. These games are run many times and the average score is taken for an accurate representation of agent performance.

1.1 Flights

Singular flights are sold to and from the vacation venue daily and there is no limit to number of flights bought. Prices are set according to a stochastic function, resulting in prices starting between \$250 and \$400, and changed every 10 seconds to a new price depending on the function. Prices will, however, always remain between \$150 and \$800. Agents cannot sell flight tickets themselves.

1.2 Hotels

Two hotels exist, one considered superior in all ways. Both hotels contain 16 rooms and hold separate auctions for rooms for each day. A random hotel auction closes every minute, with the cost each room being the 16th highest bid price in the auction. Clients cannot swap hotels during their stay and agents cannot sell hotel tickets either.

1.3 Entertainment

There exist three different types of entertainments and each agent receives 12 tickets on different days at the start of the game. Each client has a preference for different entertainment (which one they would like to see the most etc.), which allows for agents to sell and buy different tickets. However, each client can attend a certain type of entertainment only once during their stay.

1.4 Game

In a game instance, each agent has 8 clients with different preferences for arrival and departure dates and hotel and entertainment types. These agents compete with each other, making bids for flights, hotels and selling/buy entertainment tickets to get the highest utility. The utility score is calculated by:

1000 – travel_penalty + hotel_bonus + fun_bonus

Where

travel_penalty = 100*difference between flight dates and preferred dates)

hotel_bonus = client preference for the nice hotel (ranging from 50 to 150)

fun_bonus = cumulative preference values for each entertainment event the client attended (ranging from 0 to 200 for each event).

The final score is the cost subtracted from the utility.

16 instances of the game were run to allow for an average score of each agent and more accurate representation of each agents' performance.

2. CHALLENGES

There were a number of challenges faced by us while designing the agent.

2.1 Flights

The advantage of the flights auction was that there was no restrictions with regarding the number of flights available each day, meaning there is not a competitions against the other agents. However, the cost of the tickets initially is between 250 & 400, rising or declining to 150 & 800 at most. This means, we may want to think about when exactly we want to buy a ticket, so that we can take advantage of the function and get the ticket for as cheap as possible.

2.2 Hotels

Hotels are the bottleneck of the whole competition and the single limiting factor. One of the reasons for this is the limited number of hotel rooms. This means that, given the different client preferences and combinations, our agent could potentially miss out on the hotels if our hotel bids aren't good enough. However, this also means that if we miss out on a hotel room. This is the main problem, since it means that our package becomes void as the client needs to stay at a hotel and cannot switch to the other hotel. Moreover, this would mean buying another flight to accommodate the client, thereby increasing the cost of the package.

2.3 Entertainment

Entertainment allows agents to buy and sell tickets. This means that the agent can look to obtain the high utility tickets from other agents for their packages while also getting rid of unwanted tickets. However, this is a bit more complicated. We do not want to buy tickets which are not high utility, thus going into loss. Nor do we want to overpay than our utility. Lastly, we may also need to check how much we sell our tickets for. We do not want to sell a ticket too cheaply because that might mean that another agent gets a ticket for a high utility, thus increasing their own scores.

2.4 General

Finally, apart from the different issues regarding the different systems, we also need to make our agent more reactive and dynamic, changing our strategies as we go. We also want to have an idea of possible utilities we can get, and also a track of our variables and tickets we buy, therefore allowing us to better understand where we stand. Agentware code is also extremely legacy so there is a pressing need to bring Object-Oriented programming into the picture.

3. DESIGN

With the different challenges in mind, we firstly set about making the code a lot more readable. To achieve this, we created Client, ClientPackage classes to keep track of different packages we created for our clients and also checking if they are feasible or not. We also created a tracker class to keep track of different tickets that our agent had for different clients.

3.1 Flights

Most of the time, flight prices increase with time. This meant that a good strategy was to buy the ticket as soon as possible. It would allow us to get the flight tickets for a cheap price and get the flight purchase out of the way. However, it was obviously possible that sometimes, flight tickets would be more expensive at the beginning and hence cost us more in the long run. Moreover, they could also decrease in price. Therefore, we reached a sort of a compromise such that we did not always buy flights at the beginning. We would wait for a while and follow the function. If the flight prices kept increasing, we would buy the flight quickly, thus saving us from higher prices. This meant that there would be many times where we saved money on flights by not buying flights at the beginning.

3.2 Hotels

Hotels were the most important part of the auction so there were many different strategies we placed to have an overall improvement in our agent behavior.

3.2.1 Deciding on hotel

We also do not want to buy good hotels if they stretch over the holiday and are a low utility. Therefore, any client with hotel preference lower than 90 is given the bad hotel. Moreover, if their stay is over 2 day, we automatically give them the bad hotel. This means we can somewhat even out the rooms, thus reducing chance of bottlenecking somewhere.

3.2.2 Reacting on competition

The naïve agent strategy was to always add 50 to the agent asking price. This meant that the increase in agent bidding was static. Instead, we decided to calculate the different between the last two ask prices. This way, we are reacting on the competition and have a better chance of getting what we need.

3.2.3 Rescheduling

One big problem with hotels is that if we do not get a hotel room in a trip, it could potentially ruin the whole package, giving us a considerable amount of loss. To mitigate this problem, we changed the naïve plan of the dummy agent. Instead of only buying flights at the beginning, we re-calculate the feasibility of our package and buy another flight in a bid to keep complete our package. This is only done in the case that it is possible to complete the package with our new flight, meaning that we take a small hit with the extra cost of the flight, but we manage to complete feasible packages, giving us good utility.

3.2.4 Limit

Another problem is overbidding. This can be in the case where some agents end up bidding high on some tickets. Our agent would try and increase on that, but we do not want to end up spending too much money, sending us into a negative score. Therefore, we also enforce a limiting price, so that we do not end up going in loss.

3.2.5 Shot in the dark

One last thing we do is bid for a hotel in an auction that we do not need. This would only be applicable in the case that we have an empty. IN any case, our agent makes a small bid of 20 in that auction. If we get the room, we get it for a cheap price and it could possibly be used in a case where a client is coming for a stay for a single day. If we do not receive the hotel room, this means we still managed to raise the selling price of room, a win-win for us!

3.3 Entertainment

Generally, the best value for money price to sell tickets is between 80 and 60. With this in mind, we start a function to start selling at 130 initially, slowly decreasing our price over time till it costs 85, the lowest we go. This is better than the strategy of lowering till 0, which would also possibly give an advantage to the other agents. With regards to buying the tickets, we only bid as much as the utility of the ticket, so that we do not overspend.

4. ANALYSIS

Place Tables/Figures/Images in text as close to the reference as possible (see Figure 1). It may extend across both columns to a maximum width of 17.78 cm (7").

Captions should be Times New Roman 9-point bold. They should be numbered (e.g., "Table 1" or "Figure 2"), please note that the word for Table and Figure are spelled out. Figure's captions should be centered beneath the image or picture, and Table captions should be centered above the table body.

5. CONCLUSIONS

As it can be seen from the results, Agent Elman did well above average.

5.1 Subsections

The heading of subsections should be in Times New Roman 12-point bold with only the initial letters capitalized. (Note: For subsections and subsubsections, a word like *the* or *a* is not capitalized unless it is the first word of the header.)

5.1.1 Subsubsections

The heading for subsubsections should be in Times New Roman 11-point italic with initial letters capitalized and 6-points of white space above the subsubsection head.

5.1.1.1 Subsubsections

The heading for subsubsections should be in Times New Roman 11-point italic with initial letters capitalized.

5.1.1.2 Subsubsections

The heading for subsubsections should be in Times New Roman 11-point italic with initial letters capitalized.

6. ACKNOWLEDGMENTS

Our thanks to ACM SIGCHI for allowing us to modify templates they had developed.

7. REFERENCES

[1] Bowman, M., Debray, S. K., and Peterson, L. L. 1993. Reasoning about naming systems. *ACM Trans. Program. Lang. Syst.* 15, 5 (Nov. 1993), 795-825. DOI= http://doi.acm.org/10.1145/161468.16147.

[2] Ding, W. and Marchionini, G. 1997. *A Study on Video Browsing Strategies*. Technical Report. University of Maryland at College Park.

[3] Fröhlich, B. and Plate, J. 2000. The cubic mouse: a new device for three-dimensional input. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (The Hague, The Netherlands, April 01 - 06, 2000). CHI '00. ACM, New York, NY, 526-531. DOI= http://doi.acm.org/10.1145/332040.332491.

[4] Tavel, P. 2007. *Modeling and Simulation Design*. AK Peters Ltd., Natick, MA.

[5] Sannella, M. J. 1994. *Constraint Satisfaction and Debugging for Interactive User Interfaces*. Doctoral Thesis. UMI Order Number: UMI Order No. GAX95-09398., University of Washington.

[6] Forman, G. 2003. An extensive empirical study of feature selection metrics for text classification. *J. Mach. Learn. Res.* 3 (Mar. 2003), 1289-1305.

[7] Brown, L. D., Hua, H., and Gao, C. 2003. A widget framework for augmented interaction in SCAPE. In *Proceedings of the 16th Annual ACM Symposium on User Interface Software and Technology* (Vancouver, Canada, November 02 - 05, 2003). UIST '03. ACM, New York, NY, 1-10. DOI= http://doi.acm.org/10.1145/964696.964697.

[8] Yu, Y. T. and Lau, M. F. 2006. A comparison of MC/DC, MUMCUT and several other coverage criteria for logical decisions. *J. Syst. Softw.* 79, 5 (May. 2006), 577-590. DOI= http://dx.doi.org/10.1016/j.jss.2005.05.030.

[9] Spector, A. Z. 1989. Achieving application requirements. In *Distributed Systems*, S. Mullender, Ed. ACM Press Frontier Series. ACM, New York, NY, 19-33. DOI= http://doi.acm.org/10.1145/90417.90738.

Columns on Last Page Should Be Made As Close As Possible to Equal Length