CM50206 Intelligent Agents: TAC Report

Thomas Smith

Centre for Digital Entertainment
University of Bath
taes22@bath.ac.uk

Abstract. Lorem ipsum[1]

1 Introduction

[2]

2 NLG and the Semantic Web

2.1 NLG for Ontology Engineering

2.2 NLG for Publication

3 Future Work

4 Conclusions

References

[1] Greenwald, A., Lee, S.J., Naroditskiy, V.: Roxybot-06: Stochastic prediction and optimization in TAC travel. Journal of Artificial Intelligence Research 36(1), 513–546 (2009)

[2] Wellman, M.P., Reeves, D.M., Lochner, K.M.: Price prediction in a trading agent competition [extended abstract]. In: Proceedings of the 4th ACM conference on Electronic commerce. pp. 216–217. ACM (2003)

A Source Listing

Modified DummyAgent.java, licensing skipped

```
128 package se.sics.tac.aw;
129 import se.sics.tac.util.ArgEnumerator;
130
131 import java.util.ArrayList;
132 import java.util.Arrays;
133 import java.util.Iterator;
134 import java.util.List;
135 import java.util.Map.Entry;
136 import java.util.TreeMap;
137 import java.util.Map;
138 import java.util.logging.*;
139
140 public class DummyAgent extends AgentImpl {
141
142   private static final Logger log =
143     Logger.getLogger(DummyAgent.class.getName());
144
145   private static final boolean DEBUG = false;
146
147   private static final int FLIGHTS = 8;
148   private static final float FLIGHT_MIN = 150.0f;
149   private static final float FLIGHT_MAX = 800.0f;
150
151   //------------------ int z = ??;    // z per flight : bound
        on final peturbation. Unknown
152   private static final int c = 10;    // -c : lower bound of
        possible z values
153   private static final int d = 30;    //  d : upper bound for
154   private static final float T = 540.0f;  //  T : total game
        time in seconds
155
156   private float[] bidPrices;
157   private float[] currPrices;
158   private float[] flightDeltas;
159
160   private List<Map<Integer, Float>> Pz;
161   private ArrayList<int[]> clientEntPrefs;
162
163   //INVESTIGATE: this code doesn't get called between games.
164   //        (re-)initialisation of values moved to
        gameStarted()
165   protected void init(ArgEnumerator args) {
166     bidPrices = new float[agent.getAuctionNo()];
167     currPrices = new float[agent.getAuctionNo()];
168     flightDeltas = new float[FLIGHTS];
169
```

```
170     Pz = new ArrayList<Map<Integer, Float>>();
171
172   }
173
174   public void quoteUpdated(Quote quote) {
175     int auction = quote.getAuction();
176     int auctionCategory = agent.getAuctionCategory(auction);
177     if (auctionCategory == TACAgent.CAT_HOTEL) {
178       int alloc = agent.getAllocation(auction);
179       if (alloc > 0 && quote.hasHQW(agent.getBid(auction)) &&
              quote.getHQW() < alloc) {
180         Bid bid = new Bid(auction);
181         // Can not own anything in hotel auctions...
182         bidPrices[auction] = quote.getAskPrice() + 50;
183         bid.addBidPoint(alloc, bidPrices[auction]);
184         if (DEBUG) {
185           log.finest("submitting bid with alloc="
186               + agent.getAllocation(auction)
187               + " own=" + agent.getOwn(auction));
188         }
189         agent.submitBid(bid);
190       }
191     } else if (auctionCategory == TACAgent.CAT_ENTERTAINMENT)
          {
192       int alloc = agent.getAllocation(auction) - agent.getOwn
            (auction);
193       if (alloc != 0) {
194         Bid bid = new Bid(auction);
195         if (alloc < 0)
196           bidPrices[auction] = 200f - (agent.getGameTime() *
              120f) / 720000;
197         else
198           bidPrices[auction] = 50f + (agent.getGameTime() *
              100f) / 720000;
199         bid.addBidPoint(alloc, bidPrices[auction]);
200         if (DEBUG) {
201           log.finest("submitting bid with alloc="
202               + agent.getAllocation(auction)
203               + " own=" + agent.getOwn(auction));
204         }
205         agent.submitBid(bid);
206       }
207     } else if (auctionCategory == TACAgent.CAT_FLIGHT) {
208       // calculate delta from last know price (ternary guard
            for initialisation spike)
209       flightDeltas[auction] = quote.getAskPrice() - ((
            currPrices[auction] > 0.0f)? currPrices[auction] :
            quote.getAskPrice());
210       currPrices[auction] = quote.getAskPrice();
```

```
211       log.fine("got quote for auction " + auction + " with
              price " + currPrices[auction] + "( delta: " +
              flightDeltas[auction] + ")");
212     }
213   }
214
215   public void quoteUpdated(int auctionCategory) {
216     log.fine("All quotes for "
217        + agent.auctionCategoryToString(auctionCategory)
218        + " have been updated");
219     if (auctionCategory == TACAgent.CAT_FLIGHT) {
220       long seconds = agent.getGameTime();
221       log.fine("Predicting future flight minima after " +
              seconds/1000 + " seconds");
222       flight_predictions((int) (seconds/1000));
223       expected_minimum_price((int) (seconds/1000));
224       for (int i = 0; i < FLIGHTS; i++) {
225         log.fine("Flight " + i + ": current price is " +
              currPrices[i] + ", expected minimum is " +
              bidPrices[i]);
226         // if (the game is ending soon, or current price is
              within 5% of the expected minimum) and we need
              the flight and we've had time to study trends
227         if ((seconds > 500*1000 || currPrices[i] < 1.05 *
              bidPrices[i]) && agent.getAllocation(i) - agent.
              getOwn(i) > 0 && seconds > (20 + 10*i) * 1000) {
228           log.fine("Bidding.");
229           Bid bid = new Bid(i);
230           bid.addBidPoint(agent.getAllocation(i) - agent.
                getOwn(i), currPrices[i]);
231           if (DEBUG) {
232             log.fine("submitting bid with alloc=" + agent.
                  getAllocation(i)
233                + " own=" + agent.getOwn(i));
234           }
235           agent.submitBid(bid);
236         }
237       }
238     }
239   }
240
241   public void bidUpdated(Bid bid) {
242     log.finer("Bid Updated: id=" + bid.getID() + " auction="
243        + bid.getAuction() + " state="
244        + bid.getProcessingStateAsString());
245     log.finer("       Hash: " + bid.getBidHash());
246   }
247
248   public void bidRejected(Bid bid) {
249     log.warning("Bid Rejected: " + bid.getID());
```

```
250     log.warning("      Reason: " + bid.getRejectReason()
251     + " (" + bid.getRejectReasonAsString() + ')');
252   }
253
254   public void bidError(Bid bid, int status) {
255     log.warning("Bid Error in auction " + bid.getAuction() +
            ": " + status
256     + " (" + agent.commandStatusToString(status) + ')');
257   }
258
259   public void gameStarted() {
260     log.fine("Game " + agent.getGameID() + " started!");
261
262     //reinitialise prices, deltas and z-probabilities
263     bidPrices = new float[agent.getAuctionNo()];
264     currPrices = new float[agent.getAuctionNo()];
265     flightDeltas = new float[FLIGHTS];
266
267     Pz = new ArrayList<Map<Integer, Float>>();
268     // cache the value of the uniform initial probability of
            any z
269     // INVESTIGATE: is there an off-by-one error here? surely
            there are 41 z values
270     float uniformP = 1.0f / (c+d);
271     for (int flight = 0; flight < FLIGHTS; flight++) {
272       TreeMap<Integer, Float> m = new TreeMap<Integer, Float
              >();
273       for (int z = 0-c; z <= d; z++) {
274         m.put(z, uniformP);
275       }
276       Pz.add(m);
277     }
278     log.fine("Initialised variables. Pz.size(): " + Pz.size()
            + " Pz[3].size(): " + Pz.get(3).size());
279     log.fine("    Pz[3].get(-2): " + Pz.get(3).get(-2));
280
281     calculateAllocation();
282     sendBids();
283   }
284
285   public void gameStopped() {
286     log.fine("Game Stopped!");
287     for (int i = 0; i < FLIGHTS; i++) {
288       log.fine("bidPrices[" + i + "]: " + bidPrices[i] + "\t
              currPrices[" + i + "]: " + currPrices[i]);
289     }
290
291   }
292
293   public void auctionClosed(int auction) {
```

```
294       log.fine("*** Auction " + auction + " closed!");
295     }
296
297     // Nested class to represent ranges for flight value
            peturbations
298     class Range {
299
300         private float low, high;
301
302         public Range(float l, float h){
303             this.low = l;
304             this.high = h;
305         }
306
307         public boolean contains(float number){
308             return (number >= low && number <= high);
309         }
310
311         //generates a valid range given hypothetical z and t in
                seconds
312         // (c and d are constants used in the generation of z
                by the server; 10 and 30 respectively)
313         public Range(int t, int z) {
314           float x = c + (t/T)*(z-c);
315           if (x > 0) {
316             this.low = 0-c;
317             this.high = x;
318             return;
319           } else if (x < 0) {
320             this.low = x;
321             this.high = c;
322             return;
323           }
324           this.low = 0-c;
325           this.high = c;
326         }
327
328         // return the uniform probability of any int within the
                range
329         public float uniformP() {
330           return 1.0f / (high - low);
331         }
332
333         // return the midpoint of the range, used for expected
                values
334         public float getMid() {
335           return (high - low) / 2.0f;
336         }
337
338         public String toString() {
```

```
339         return "(" + this.low + ") - (" + this.high + ")";
340       }
341
342   }
343
344   // for each possible value of z for each flight, calculate
          the likelihood that that value
345   //  is the one the server is using to generate the prices
346   private void flight_predictions(int t) {
347     int flightNo = 0;
348     // for each flight (each initialised with possible values
           of z from -c to d [-10,30])
349     for (Map<Integer, Float> flight : Pz) {
350
351       log.fine("Calculating for flight " + flightNo + "; " +
            flight.size() + " values for z remain.");
352       float runningTotal = 0;
353
354       Iterator<Entry<Integer, Float>> z = flight.entrySet().
            iterator();
355       Range r;
356
357       // for each remaining possible value of z
358       while (z.hasNext()) {
359         Entry<Integer, Float> p = (Entry<Integer, Float>)z.
              next();
360         r = new Range(t, p.getKey());   // calculate the
              range of possible values for y
361         if ( r.contains(flightDeltas[flightNo]) ) {
                    // if y is within range for this z
362           p.setValue( r.uniformP() * p.getValue());
363           runningTotal += p.getValue();
364         } else {
365           log.finest("" + currPrices[flightNo] + " is outside
                probable range: " + flightDeltas[flightNo] + "
                exceeds " + r.toString());
366           z.remove(); //this value of z cannot explain
                observed prices, discard it.
367         }
368       }
369
370       // normalise the probablilities of each z value
            remaining plausible for this flight
371       for (Entry<Integer, Float> p : flight.entrySet()) {
372         flight.put(p.getKey(), p.getValue()/runningTotal);
373       }
374
375       flightNo++;
376     }
377
```

```
378     return;
379   }
380
381   // for each possible value of z for each flight, calculate
          the minima along an expected walk
382   //  take a weighted average of these minima according to
          the probabilites of z
383   private void expected_minimum_price(int t) {
384     int flightNo = 0;
385     // for each flight
386     for (Map<Integer, Float> flight : Pz) {
387
388       float runningTotal = 0.0f;
389       //for each plausible value of z
390       for (Map.Entry<Integer, Float> z : flight.entrySet()) {
391         float min = Float.POSITIVE_INFINITY;
392         float p = currPrices[flightNo]; //current price for
              this flight
393         //simulate forwards to the end of the game
394         for (int tau = t; tau <= T; tau+=10) {
395           //peturbing by naive expectations of delta
396           float delta = new Range(tau, z.getKey()).getMid();
397           p = Math.max(FLIGHT_MIN, Math.min(FLIGHT_MAX, p +
                delta ));
398           //track the minimum price observed
399           if (p < min) {
400             min = p;
401           }
402         }
403         // multiply min by the probability that this is the
              one true z
404         runningTotal += min * z.getValue();
405       }
406       // set our expected minimum for this flight to the
            weighted average
407       bidPrices[flightNo] = runningTotal;
408       flightNo++;
409     }
410
411   }
412
413   private void sendBids() {
414     for (int i = 0, n = agent.getAuctionNo(); i < n; i++) {
415       int alloc = agent.getAllocation(i) - agent.getOwn(i);
416       float price = -1f;
417       switch (agent.getAuctionCategory(i)) {
418       case TACAgent.CAT_FLIGHT:
419         // don't bid on flights at the start of the game - we
              wait for the opportune moment
420         break;
```

```
421       case TACAgent.CAT_HOTEL:
422         if (alloc > 0) {
423           price = 200;
424           bidPrices[i] = 200f;
425         }
426         break;
427       case TACAgent.CAT_ENTERTAINMENT:
428         if (alloc < 0) {
429           price = 200;
430           bidPrices[i] = 200f;
431         } else if (alloc > 0) {
432           price = 50;
433           bidPrices[i] = 50f;
434         }
435         break;
436       default:
437         break;
438       }
439       if (price > 0) {
440         Bid bid = new Bid(i);
441         bid.addBidPoint(alloc, price);
442         if (DEBUG) {
443           log.finest("submitting bid with alloc=" + agent.
                getAllocation(i)
444                 + " own=" + agent.getOwn(i));
445         }
446         agent.submitBid(bid);
447       }
448     }
449   }
450 
451   //store the client preferences as allocated desires in
        particular auctions.
452   private void calculateAllocation() {
453     clientEntPrefs = new ArrayList<int[]>();
454 
455     for (int i = 0; i < 8; i++) {
456       int inFlight = agent.getClientPreference(i, TACAgent.
            ARRIVAL);
457       int outFlight = agent.getClientPreference(i, TACAgent.
            DEPARTURE);
458       int hotel = agent.getClientPreference(i, TACAgent.
            HOTEL_VALUE);
459       int type;
460 
461       // Get the flight preferences auction and remember that
             we are
462       // going to buy tickets for these days. (inflight=1,
            outflight=0)
```

```
463       int auction = agent.getAuctionFor(TACAgent.CAT_FLIGHT,
              TACAgent.TYPE_INFLIGHT, inFlight);
464       agent.setAllocation(auction, agent.getAllocation(
              auction) + 1);
465       auction = agent.getAuctionFor(TACAgent.CAT_FLIGHT,
              TACAgent.TYPE_OUTFLIGHT, outFlight);
466       agent.setAllocation(auction, agent.getAllocation(
              auction) + 1);
467
468       // if the hotel value is greater than 70 we will select
               the
469       // expensive hotel (type = 1)
470       if (hotel > 70) {
471         type = TACAgent.TYPE_GOOD_HOTEL;
472       } else {
473         type = TACAgent.TYPE_CHEAP_HOTEL;
474       }
475       // allocate a hotel night for each day that the agent
              stays
476       for (int d = inFlight; d < outFlight; d++) {
477         auction = agent.getAuctionFor(TACAgent.CAT_HOTEL,
                type, d);
478         log.finer("Adding hotel for day: " + d + " on " +
                auction);
479         agent.setAllocation(auction, agent.getAllocation(
                auction) + 1);
480       }
481
482       //calculate the client's ordered preferences
483       clientEntPrefs.add(getClientEntPrefs(i));
484       //allocate them their first choice - from what we own
              if possible
485       bestEntDay(inFlight, outFlight, i, 0);
486
487     }
488
489     //loop through all of the clients, allocating them their
          second and third preferences if possible
490     for (int pref = 1; pref <= 2; pref++) {
491       for (int client = 0; client < 8; client++) {
492         bestEntDay(agent.getClientPreference(client, TACAgent
                .ARRIVAL), agent.getClientPreference(client,
                TACAgent.DEPARTURE), client, pref);
493       }
494     }
495   }
496
497   private void bestEntDay(int inFlight, int outFlight, int
        client, int pref) {
498     //retrieve the type of entertainment we're looking for
```

```
499     int type = clientEntPrefs.get(client)[pref];
500     for (int i = inFlight; i < outFlight; i++) {
501       //skip this date if the client already has allocated
              entertainment
502       if (0-i == clientEntPrefs.get(client)[0] || 0-i ==
              clientEntPrefs.get(client)[1]) {
503         continue;
504       }
505       int auction = agent.getAuctionFor(TACAgent.
              CAT_ENTERTAINMENT, type, i);
506       if (agent.getAllocation(auction) < agent.getOwn(auction
              )) {
507         log.finer("Adding entertainment " + type + " on " +
                auction);
508         agent.setAllocation(auction, agent.getAllocation(
                auction) + 1);
509         //double up on the prefs to store which days are
                already allocated
510         clientEntPrefs.get(client)[pref] = -i;
511         return;
512       }
513     }
514 
515     // If none left and needy, just take the first...
516     if (pref == 0) {
517       int auction = agent.getAuctionFor(TACAgent.
              CAT_ENTERTAINMENT, type, inFlight);
518       agent.setAllocation(auction, agent.getAllocation(
              auction) + 1);
519       clientEntPrefs.get(client)[0] = -inFlight;
520       return;
521     }
522   }
523 
524   // return a short ordered list for the order of client
        entertainment type preferences
525   private int[] getClientEntPrefs(int client) {
526     int e1 = agent.getClientPreference(client, TACAgent.E1);
527     int e2 = agent.getClientPreference(client, TACAgent.E2);
528     int e3 = agent.getClientPreference(client, TACAgent.E3);
529 
530     int orderedPrefs[] = {0,0,0};
531 
532     orderedPrefs[0] = (e1 > e2 && e1 > e3)? TACAgent.
            TYPE_ALLIGATOR_WRESTLING : (e2 > e3)? TACAgent.
            TYPE_AMUSEMENT : TACAgent.TYPE_MUSEUM;
533     orderedPrefs[1] = (e1 < Math.max(e1, Math.max(e2, e3)) &&
             e1 > Math.min(e1, Math.min(e2, e3)))? TACAgent.
            TYPE_ALLIGATOR_WRESTLING : (e2 < Math.max(e1, Math.
```

```
          max(e2, e3)) && e2 > Math.min(e1, Math.min(e2, e3)))?
           TACAgent.TYPE_AMUSEMENT : TACAgent.TYPE_MUSEUM;
534     orderedPrefs[2] = (e1 < e2 && e1 < e3)? TACAgent.
          TYPE_ALLIGATOR_WRESTLING : (e2 < e3)? TACAgent.
          TYPE_AMUSEMENT : TACAgent.TYPE_MUSEUM;
535     log.fine("client " + client + ": " + orderedPrefs[0] + "
          " + orderedPrefs[1] + " " + orderedPrefs[2]);
536     return orderedPrefs;
537   }
538
539
540
541   //
          ----------------------------------------------------------------------

542   // Only for backward compability
543   //
          ----------------------------------------------------------------------

544
545   public static void main (String[] args) {
546     TACAgent.main(args);
547   }
548
549 } // DummyAgent
```