

# ANAC17 Agent implementations

Γιώργος Σταματάκης

AM: 2013 030 154

Πολυπρακτορικά Συστήματα [ΠΛΗ517]

[github.com/gstamatakis/ANAC-agents](https://github.com/gstamatakis/ANAC-agents)

## Περίληψη

Στα πλαίσια του μαθήματος Πολυπρακτορικά Συστήματα [ΠΛΗ517] και του διαγωνισμού ANAC17 υλοποιήθηκαν 2 πράκτορες με διαφορετικές στρατηγικές. Σκοπός ήταν να γίνει η επιλογή του καλύτερου για να συμμετάσχει στο διαγωνισμό ANAC. Οι πράκτορες αναπτύχθηκαν στο περιβάλλον GENIUS 7.1.3 με σκοπό τη συμμετοχή τους στην κατηγορία multi-lateral SAOP negotiations.

Όσον αφορά την υλοποίησή του ο κεντρικός πράκτορας αποτελείται από αρκετά modules που το καθένα λειτουργεί αυτόνομα με αποτέλεσμα να διευκολύνεται η επέκταση και η μίξη των πρακτόρων. Με λίγα λόγια επαναχρησιμοποιείται μεγάλο μέρος του κώδικα ανάμεσα στους 2 πράκτορες που υλοποιήθηκαν (κυρίως αλγόριθμοι αναζήτησης και δομές αποθήκευσης) ενώ η χρήση ενός ενιαίου API για όλους τους πράκτορες επιτρέπει την μίξη των στοιχείων των πρακτόρων πχ. Αν υπάρχει ένας A τρόπος αναζήτησης του search space μας και ένας B άλλος τρόπος μπορούμε εύκολα να διαπιστώσουμε ποιος από τους 2 είναι καλύτερος για κάθε πράκτορα. Αυτό το modular design στο τέλος μας έδωσε έναν hybrid πράκτορα που χρησιμοποιεί στρατηγικές και τεχνάσματα και από τους 2 πράκτορες. Τέλος η εναλλαγή στρατηγικών κατά τη διάρκεια του διαγωνισμού είναι σχεδόν απαραίτητη μιας και οι πράκτορες τα τελευταία χρόνια χρησιμοποιούν data peristency για να αναλύσουν τη στρατηγική του αντιπάλου με αποτέλεσμα η εναλλαγή στρατηγικών να είναι πρακτικά απαραίτητη.

Αυτή η αναφορά σπάει σε 3 βασικά μέρη, την θεωρητική προσέγγιση της στρατηγικής των πρακτόρων, την υλοποίησή τους που περιλαμβάνει την επιλογή και δημιουργία των αλγορίθμων του πράκτορα και τέλος τα πειράματα που έγιναν για την επιλογή των βέλτιστων στοιχείων του πράκτορά μας (στρατηγική, αλγόριθμοι και διάφορες σταθερές). Αν δεν αναφέρεται το domain είναι το partydomain με partyutilities σε αύξοντα αριθμό ανάλογα με τον αριθμό των πρακτόρων.

## Πράκτορες

### I. Time Agent

Πράκτορας που βασίζεται στην πρόταση του βιβλίου Multi-agent and Complex Systems στο κεφάλαιο Preliminary Estimating Method of Opponent's Preferences Using Simple Weighted Functions for Multilateral Multi-issue Negotiations. Έγιναν κάποιες αλλαγές στον πράκτορα και προστέθηκε μνήμη για καλύτερη επιλογή μερικών σταθερών.

### II. Threshold Agent

Πράκτορας που βασίζεται στη στρατηγική του πράκτορα FARMA με πολλές αλλαγές σε διάφορα δομικά του μέρη. Ο καινούριος πράκτορας πετυχαίνει καλύτερα αποτελέσματα και νικάει τον παλιότερο πράκτορα Farma Agent.

## Θεωρητική προσέγγιση πρακτόρων

### Time Agent

Ο πράκτορας αυτός χαρακτηρίζεται από μια μέθοδο εκτίμησης που χρησιμοποιεί απλές σταθμισμένες συναρτήσεις μετρώντας της αξίας αξιολόγησης (evaluation value) του αντιπάλου για κάθε issue. Επειδή οι διαπραγματεύσεις είναι multi-lateral όταν ο πράκτορας υπολογίσει κάποια partial utility functions θα τα σταθμίσει κατάλληλα και θα τα συνοψίσει σε μία τελική utility function. Ουσιαστικά ο πράκτορας θα προσεγγίζει, μέσω κάποιας στρατηγικής, το utility function των αντιπάλων του αθροίζοντας με βάρη την τιμή κάθε issue.

Όσον αφορά τη λογική του πράκτορα έστω ένα σύνολο από Bid  $s_1 \dots s_N$  για N issues και οι αντίστοιχες τιμές τους στο utility space  $eval(s_i)$ . Αναθέτουμε σε κάθε issue μια κανονικοποιημένη τιμή βάρους  $w_i$ ,  $\sum_{i \in I} w_i = 1$  στο utility space. Ακολουθεί το utility

$$U(\mathbf{v}) = \sum_{i=1}^N w_i \cdot eval(s_i).$$

function όλων των Bid.

Όσον αφορά την προκαθορισμένη διάρκεια των διαπραγματεύσεων (deadline) κοινοποιείται σε ένα χρονικό διάστημα  $t \in [0, 1]$  όπου 0 είναι η αρχή και 1 το τέλος των διαπραγματεύσεων. Επειδή όμως υπάρχουν σε αρκετές διαπραγματεύσεις discount factors, που ορίζονται σε διάστημα  $[0, 1]$ , πρέπει όταν είναι απαραίτητο να υπολογίζεται το discounted utility του outcome  $\omega$  από το undiscounted utility  $U$ . Ο παρακάτω τύπος δίνει το ζητούμενο utility όταν δίνεται ο κανονικοποιημένος χρόνος  $t$ .

$$U_D^t(\omega) = U(\omega) \cdot d^t$$

Μπορεί κανείς να παρατηρήσει ότι για  $d = 1$  (undiscounted scenario) τα 2 utilities ταυτίζονται.

Ορίζω ως objective functions του TimeAgent τις ακόλουθες συναρτήσεις

- **Objective Function 1 (Maximizing Individual Welfare):**

$$\max_s U_a(s)$$

• **Objective Function 2 (Maximizing Social Welfare):**

$$\max_s \sum_{i=1}^N U_{A_i}(s)$$

Ο πράκτορας, με άλλα λόγια, προσπαθεί να παίξει πράγματα που μεγιστοποιούν την κοινωνική ευημερία (social welfare) , δηλαδή, τα total utilities για όλους τους Agents. Τέτοια bids, εξ ορισμού, θα είναι επίσης και Pareto optimal. Ταυτόχρονα, κάθε πράκτορας προσπαθεί να παίξει πράγματα όπου η ατομική ευημερία υπερβαίνει την τιμή κράτησης (reservation value).

Στη συνέχεια γίνεται αναφορά για τη στρατηγική του πράκτορα και τον τρόπο με τον οποίο γίνεται η πρόβλεψη των utility functions. Η κεντρική ιδέα είναι ότι θα αποθηκεύουμε προσωρινά τα Bids των αντιπάλων μας και μέσω κάποιων στατιστικών που θα εξαγάγουμε από αυτά θα αποφασίζουμε για το ποιο είναι το καλύτερο response στα Bids που δεχόμαστε. Ορίζω ως  $A_N$  τον πράκτορά μας και  $a = \{A_1, A_2, \dots, A_{N-1}\}$  τους N-1 αντιπάλους του. Τα Bids του πράκτορα  $a$  αναπαριστώνται με  $B_a$  και το estimated utility function του πράκτορα  $a$

$$eval'_a(s_i) = \sum_{s' \in B_a} Boolean(s_i, s') \cdot w(s').$$

αναπαριστάται με  $eval'_a()$  που ορίζεται ως:

Να σημειωθεί ότι η συνάρτηση  $Boolean(s_i, s')$  επιστρέφει true μόνο όταν το bid  $s'$  εμπεριέχει το  $s_i$  και false διαφορετικά. Η  $w(s)$  είναι η συνάρτηση στάθμης που δείχνει τη επηρεάζει τα bids που δέχτηκε ο πράκτοράς μας. Άρα η συνάρτηση για το estimated utility είναι η ακόλουθη:

$$u_a(s) = \sum_{i=1}^N eval'_a(s_i).$$

Όπου  $u_a(s)$  η ακόλουθη συνάρτηση:

Με την 1η συνάρτηση ο πράκτορας μπορεί να πάρει το κανονικοποιημένο estimated utility σε διάστημα [0, 1]. Επιπλέον χρησιμοποιούνται οι 3 παρακάτω σταθμικές συναρτήσεις:

- Constant :  $w(s) = 1$
- Monotonically Increasing :  $w(s) = time_a(s)$

- Monotonically Decreasing :  $w(s) = 1 - time_a(s)$

Όπου η συνάρτηση  $time_a(s)$  επιστρέφει τη κανονικοποιημένη χρονική τιμή όταν ο πράκτορας α προτείνει ένα bid  $s$ .

Η στρατηγική του πράκτορα πρέπει να κρίνει με ένα έξυπνο και ορθολογικό τρόπο τα bids που πρέπει να κάνει, να δεχθεί και να απορρίψει. Αρχικά ο πράκτοράς μας έχει  $h_n$  συναρτήσεις (μία για κάθε παίκτη) που χρησιμοποιεί για να κρίνει κάθε παίκτη. Ακολουθεί η συνάρτηση  $U_{op}$  που συγκεντρώνει τις μερικές αποφάσεις για όλους τους πράκτορες και

$$U_{op}(s) = \sum_{n=1}^N h_n U_{A_n}(s).$$

χρησιμοποιείται για να πάρει το τελικό αποτέλεσμα.

Να σημειωθεί ότι αυτός ο πράκτορας θα έπαιζε αποτελεσματικά τόσο για 3 παίκτες (βλ. Κανόνες ANAC) όσο και για N παίκτες.

Οι κινήσεις του πράκτορά μας (Accept, Offer, Refuse) αποφασίζονται από τα τις παρακάτω

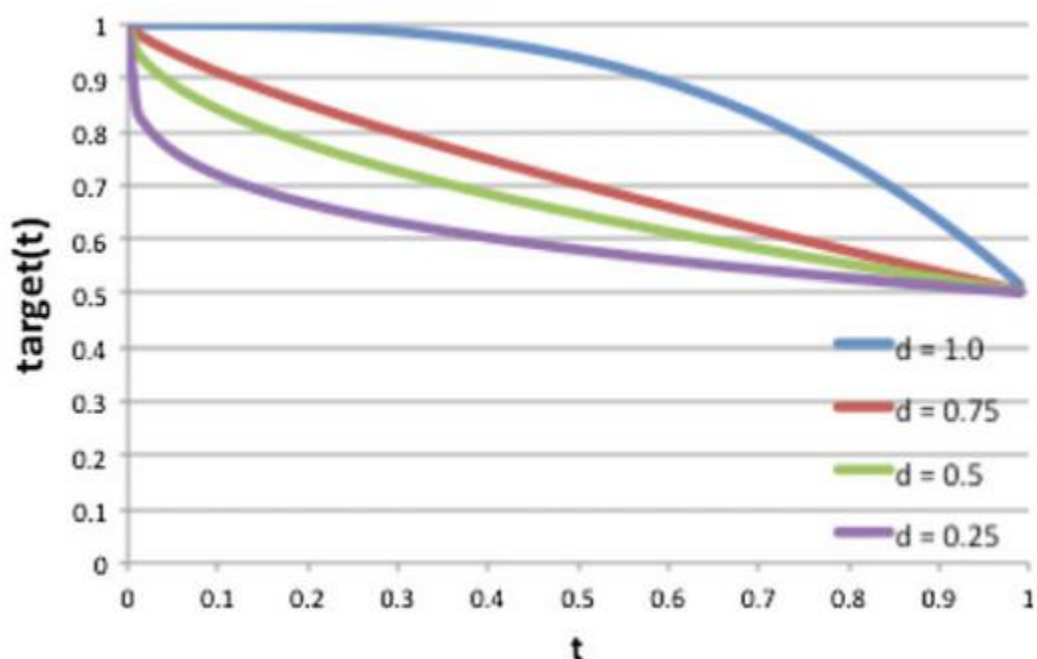
$$target_{end} = \frac{\sum_{n=1}^N h_n U_{my}(\arg \max U_{A_n})}{\sum_{n=1}^N h_n}$$

$$target(t) = \begin{cases} (1 - t^3)(1 - target_{end}) + target_{end} & (d = 1) \\ (1 - t^d)(1 - target_{end}) + target_{end} & (otherwise) \end{cases}$$

συναρτήσεις.

Η 1<sup>η</sup> συνάρτηση υπολογίζει το σταθμικό μέσο της εκτιμώμενης τιμής που προτείνει ο αντίπαλος στη τελική φάση και τον διαιρεί κατά  $h_n$ . Ο πράκτοράς μας στη συνέχεια προτείνει ένα Bid με utility που να ξεπερνά τόσο το  $target(t)$  όσο και το μεγαλύτερο  $U_{op}$ . Επειδή αυτά είναι τα Bids που θεωρεί αποδεκτά ο πράκτοράς μας θα αποδεχθεί όλα τα Bids που ξεπερνούν σε δικό του utility τις παραπάνω τιμές.

Στο διάγραμμα που ακολουθεί φαίνονται τα κατώφλια του πράκτορά μας για  $target_{end}=0.5$ .



Μπορούμε εύκολα να δούμε ότι για μικρές τιμές στο discount factor ο πράκτοράς μας φτάνει γρήγορα σε κάποιο συμβιβασμό μιας και ρίχνει απότομα το στόχο του. Κάτι τέτοιο δεν συμβαίνει για μεγάλες τιμές του discount factor που ο πράκτοράς μας αρχικά ρίχνει πολύ αργά το στόχο του ενώ στη συνέχεια προσπαθεί να καταλήξει γρήγορα σε κάποια συμφωνία. Το  $\text{target}_{\text{end}}$  μπορεί να εύκολα να δει κανείς ότι είναι το σημείο που συγκλίνουν όλες οι καμπύλες ανεξαρτήτως discount factor κάτι που διευκολύνει το χειρισμό του concession.

## Thresh Agent

Αντλώντας κάποιες ιδέες και αλγορίθμους από τον πράκτορα FARMA δημιουργήθηκε ο πράκτορας Thresh. Οι πληροφορίες που αντλήθηκαν περιλαμβάνουν μόνο τον τύπο που υπολογίζει το displacement του αντιπάλου και τον υπολογισμό του κατωφλίου ανά πάσα στιγμή. Ακολουθεί η λογική και οι αλγόριθμοι του πράκτορα.

Η στρατηγική του πράκτορα ξεκινάει συγκρίνοντας την προσφορά του αντιπάλου του με το αποτέλεσμα της συνάρτησης που υπολογίζει το αποδεκτό threshold και αν βρίσκεται εντός ορίων την αποδέχεται, διαφορετικά την απορρίπτει και προχωράει σε counter-offer ή σταματάει εντελώς τις διαπραγματεύσεις αν το αποτέλεσμα της συνάρτησης που υπολογίζει αν θα συνεχίσει τις διαπραγματεύσεις αποφασίσει θετικά.

Ξεκινώντας με τον υπολογισμό του threshold πρέπει να υπολογιστούν κάποια στατιστικά. Κάθε bid που δέχεται ο πράκτοράς μας συγκρίνεται με αυτό το χρονικά μεταβαλλόμενο κατώφλι που υπολογίζεται από τους παρακάτω τύπους.

Το προσωρινό threshold προκύπτει από το αποτέλεσμα της παρακάτω συνάρτησης, όπου το

$$\begin{cases} \min(threshold, 1 - (1 - emax) * time^2) & 1.0 - discountFactor < CutoffVal \\ \max(threshold - time, emax) & else \end{cases}$$

Για τον υπολογισμό της ποσότητας emax χρησιμοποιώ τον ακόλουθο τύπο έχοντας αρχικοποιήσει την ποσότητα αρχικά στη μονάδα. Επίσης στην περίπτωση όπου το reservation value είναι μεγαλύτερο από το emax επιλέγω αυτό σαν emax. Η ποσότητα CutoffVal είναι μια αυθαίρετα ορισμένη ποσότητα μετά την οποία θεωρούμε ότι το αποτέλεσμά μας είναι πρακτικά 0 πχ.  $10^{-5}$ ). Η μεταβλητή time καθ όλη την έκταση του κώδικά μου αναπαριστά την τωρινή χρονική στιγμή των διαπραγματεύσεων, επίσης υπάρχει και ο discount factor όπου δυσκολεύει τις διαπραγματεύσεις μας μιας και απαιτεί να παίρνουμε πιο γρήγορες αποφάσεις ειδικά όταν μειώνει το utility μας με γρήγορο ρυθμό.

$$emax \leftarrow \min(emax, avg + (1 - avg) * calWidth(avg, sd))$$

Η συνάρτηση calWidth χρησιμοποιείται για την εκτίμηση του πλάτους μετατόπισης του αντιπάλου με βάση τα στατιστικά που κρατάμε. Το αποτέλεσμα της δίνεται από τον ακόλουθο τύπο.

$$\begin{cases} sd * \sqrt{\frac{3}{avg - (avg)^2}} & avg \in [0.1, 0.9] \\ sd * \sqrt{12} & else \end{cases}$$

Το τελικό αποτέλεσμα για το κατώφλι δίνεται από τον παρακάτω τύπο και αυτό που κάνει ουσιαστικά είναι να προσφέρει ένα καλύτερο threshold όταν οι διαπραγματεύσεις βρίσκονται προς στο τέλος. Έχουμε ορίσει ένα soft και ένα hard concession time threshold (~0.9 και ~0.98



αντίστοιχα) με σκοπό την επιτάχυνση των διαπραγματεύσεων μετά το πέρας κάποιων χρονικών στιγμών.

$$\begin{cases} \min(threshold, BestOfferedUtils) & softThr < t < hardThr \\ \max(\min(threshold, BestOfferedUtils), resVal) & t > hardThr \\ threshold & else \end{cases}$$

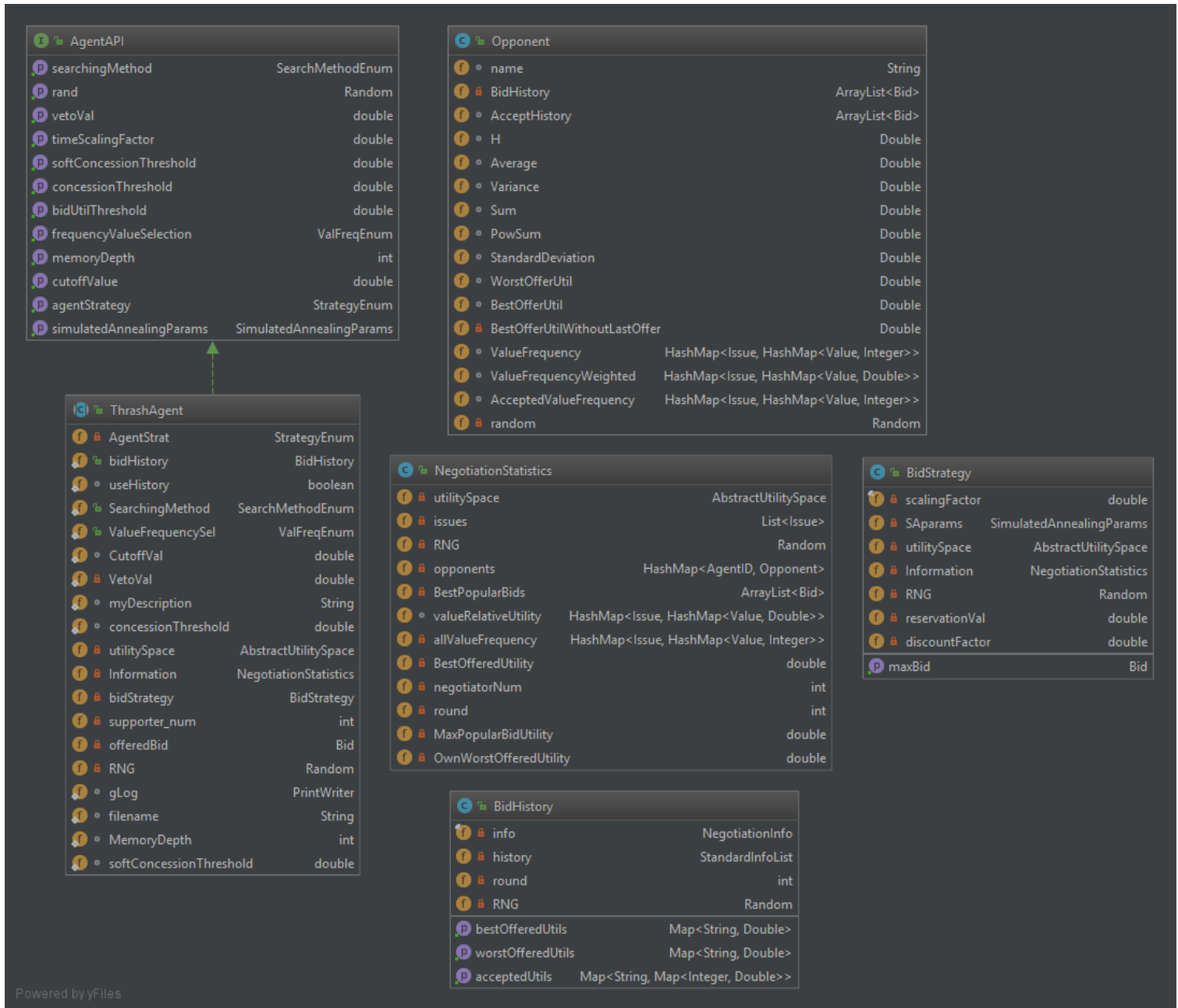
Η μεταβλητή *resVal* αντιστοιχεί στο reservation value ενώ τα *BestOfferedUtils* είναι τα καλύτερα offeres, από άποψη utility, που έχει λάβει ο πράκτοράς μας μέχρι αυτή τη στιγμή ενώ σε περίπτωση που επιτρέπεται η χρήση μνήμης (STANDARD data percistency) ο πράκτορας θα κοιτάξει για όλα τα καλύτερα offers που του έγιναν σε ένα βάθος χρόνου που καθορίζεται από το χρήστη (συνήθως 3 με 4 γύρους). Να σημειωθεί ότι αυτό συμβαίνει μόνο κατά τον 1ο κλάδο της παραπάνω συνάρτησης και μόνο όταν είναι δυνατή η χρήση persistent τιμών, διαφορετικά ο 1ος και ο 3ος κλάδος κάνουν collapse σε έναν, επίσης οι τιμές *softThr* και *hardThr* πρέπει να επιλέγονται με προσοχή μιας και μικρά concession thresholds επιτρέπουν στον πράκτορά μας να “χειραγωγηθεί” εύκολα από άλλους.

Στην τελευταία ενότητα της αναφοράς που υπάρχουν τα πειράματα μπορεί να δει κανείς ότι αυτή η στρατηγική είναι ελαφρώς καλύτερη από τη στρατηγική με τις καμπύλες και το χρονικό στόχο που είδαμε πριν, μιας και απαιτεί τόσο απλούστερους υπολογισμούς όσο και λιγότερους βρόχους επανάληψης. Τέλος να σημειωθεί ότι τα μόνα δεδομένα που αντλήθηκαν από το πράκτορα FARMA είναι μόνο οι 3 πρώτοι τύποι (*threshold*, *calWidth*, *emax*) ενώ οι αλλαγές που έκανα τόσο σε κάποιες σταθερές και παραμέτρους όσο και στην αλλαγή κάποιων αλγορίθμων (πχ. Απλοποίηση μαθηματικών εκφράσεων, βελτίωση βρόχων ,προσθήκη concession thresholds και χρήση μνήμης) βελτίωσαν τον πράκτορα με αποτέλεσμα να νικάει τον πρωτότυπο αλλά και αρκετούς άλλους πράκτορες.



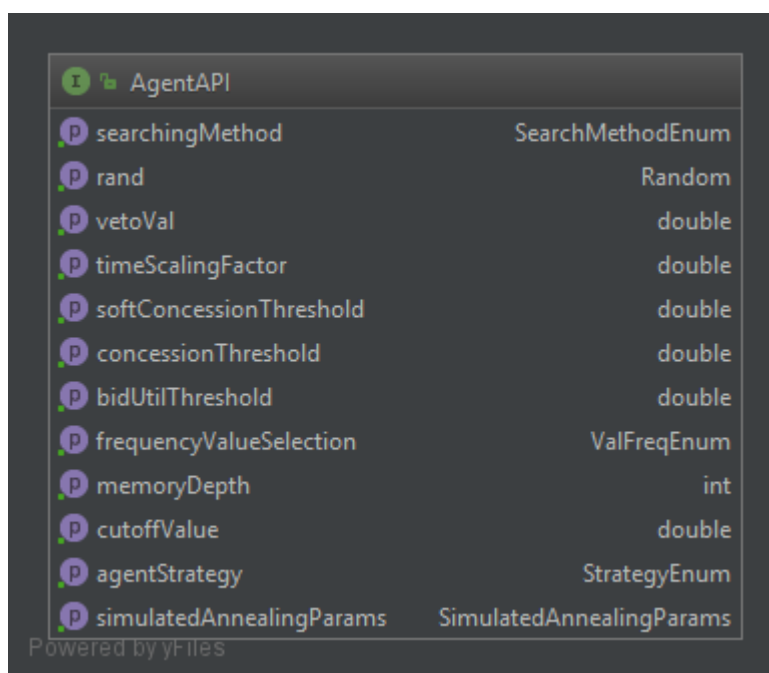
## Υλοποίηση ενιαίου πράκτορα

Ο ενιαίος πράκτορας αποτελείται από 5 βασικά modules ,τα οποία χειρίζονται ένα βασικό και αναπόσπαστο κομμάτι του, και ένα εξωτερικό interface που είναι το API του πράκτορα και προσφέρει έναν γρήγορο τρόπο για customisation. Ακολουθεί το class diagram του πράκτορα.



Συνοπτικά, το Bid History module αντλεί πληροφορίες από παλιότερες διαπραγματεύσεις εφόσον το επιτρέπει η πλατφόρμα GENIUS. Ενδεικτικά μπορεί να δώσει πληροφορίες για τις καλύτερες προσφορές που έκανε ένας από τους αντιπάλους σε προηγούμενες διαπραγματεύσεις του ίδιου τουρνουά. Ένα άλλο module είναι το BidStrategy που περιέχει τις στρατηγικές που υποστηρίζει ο πράκτορας και υλοποιείται με βάση τις συναρτήσεις που περιγράφηκαν στο προηγούμενο κεφάλαιο. Υπάρχουν επίσης και τα modules Opponent και Negotiation Statistics που κρατάνε διάφορα στατιστικά για τους αντιπάλους (πχ. average, std deviation, accepted offers) αλλά και κάποιους αλγόριθμους αναζήτησης αυτών των δεδομένων. Τέλος, υπάρχει και η abstract κλάση ThrashAgent που επεκτείνει την κλάση Agent και ενθυλακώνει τα παραπάνω modules, άρα σε περίπτωση που επιθυμεί κάποιος να προσαρμόσει μια νέα στρατηγική ή γενικότερα να επεκτείνει τη λειτουργικότητα του πράκτορα το μόνο που απαιτείται είναι να επεκτείνει αυτόν τον πράκτορα.

Για τη δημιουργία ενός τέτοιου πράκτορα πρέπει να υλοποιηθεί το παρακάτω interface οι τιμές του οποίου αναλύονται στη συνέχεια (εξαιρούνται οι rand και vetoVal, περισσότερα στην επόμενη ενότητα).



AgentAPI	
searchingMethod	SearchMethodEnum
rand	Random
vetoVal	double
timeScalingFactor	double
softConcessionThreshold	double
concessionThreshold	double
bidUtilThreshold	double
frequencyValueSelection	ValFreqEnum
memoryDepth	int
cutoffValue	double
agentStrategy	StrategyEnum
simulatedAnnealingParams	SimulatedAnnealingParams

Powered by yFiles

**searchingMethod:** Η μέθοδος με την οποία γίνεται η εξερεύνηση του search space όπως για παράδειγμα τα utilities των Bids των αντιπάλων. Η πρώτη επιλογή είναι η χρήση Simulated Annealing (SA), ένας metaheuristic αλγόριθμος που χρησιμοποιείται για την εύρεση ολικού μεγίστου μιας συνάρτησης σε περιορισμένο χρόνο, κάτι που τον κάνει ιδανικό για Agents, στην περίπτωση του πράκτορα μας χρησιμοποιείται για την εύρεση ενός optimal Bid σε περιορισμένο χρονικό διάστημα. Ο αλγόριθμος ουσιαστικά κάνει σε κάθε iteration ένα ή περισσότερα random steps ψάχνοντας κάθε στιγμή ένα καλύτερο Bid με τη θερμοκρασία του να μειώνεται κάθε φορά, κάτι που έχει ως αποτέλεσμα με το πέρασμα του χρόνου να γίνονται πιο δύσκολα δεκτά

καινούρια Bids. Να σημειωθεί όταν πέσει κάτω από ένα κατώφλι η θερμοκρασία τότε σταματάει ο αλγόριθμος και επιστρέφεται τυχαία ένα από τα Bids που έχουν επιλεγεί κάτι που μπορεί να μην είναι βέλτιστο αλλά “αρκετά καλό”. Ακολουθεί ο ψευδοκώδικας του αλγορίθμου επιλογής βέλτιστου Bid για θερμοκρασία που δεν ξεπερνάει την τιμή threshold την χρονική στιγμή time.

---

**Algorithm 1** searchingMethod

---

```

1: function SIMULATEDANNEALING(threshold, baseBid, time, issues, startTemp, endTemp, cool, steps)
2:   targetBids  $\leftarrow \emptyset$ 
3:   currentBidUtil  $\leftarrow$  getUtilityWithDiscount(baseBid, 0)
4:   curTemp  $\leftarrow$  startTemp
5:   targetBidUtil  $\leftarrow$  0
6:   nextBidUtil  $\leftarrow \emptyset$ 
7:   while curTemp > endTemp do
8:     for i = 0 to steps do
9:       issue  $\leftarrow$  random issue
10:      values  $\leftarrow$  issue.values()
11:      valueIndex  $\leftarrow$  random value between [0, values.size)
12:      nextBid  $\leftarrow$  baseBid.putValue(issue.getNumber(), values.get(valueIndex))
13:      nextBidUtil  $\leftarrow$  nextBid.discountedUtility(time)
14:      if maxBid =  $\emptyset$  or nextBidUtil  $\geq$  maxBid.discountedUtility(time) then
15:        maxBid  $\leftarrow$  nextBid
16:      end if
17:    end for
18:    newEnergy  $\leftarrow$  |threshold - nextBidUtil|
19:    curEnergy  $\leftarrow$  |threshold - currentBidUtil|
20:    p  $\leftarrow e^{-\frac{|newEnergy - curEnergy|}{curTemp}}$ 
21:    if newEnergy < curEnergy or p > random number in [0,1] then
22:      baseBid  $\leftarrow$  nextBid
23:      currentBidUtil  $\leftarrow$  nextBidUtil
24:    end if
25:    if currentBidUtil  $\geq$  threshold then
26:      if targetBids is not empty then
27:        if currentBidUtil < targetBidUtil then
28:          targetBids.clear()
29:          targetBids.add(baseBid)
30:          targetBidUtil  $\leftarrow$  currentBidUtil
31:        else if currentBidUtil = targetBidUtil then
32:          targetBids.add(baseBid)
33:        end if
34:      else
35:        targetBids.add(baseBid)
36:        targetBidUtil  $\leftarrow$  currentBidUtil
37:      end if
38:    end if
39:    curTemp  $\leftarrow$  curTemp * cool
40:  end while
41:  return targetBids
42: end function

```

---

Αντί για SA μπορεί επίσης να χρησιμοποιηθεί και relative utility search με τον παρακάτω αλγόριθμο για το ίδιο ακριβώς ζητούμενο. Η ιδέα πίσω από τον αλγόριθμο είναι ότι ψάχνοντας τις τιμές των τυχαίων issues για κάθε τιμή ενός issue τη συγκρίνω με προηγούμενες τιμές που έχω κρατήσει στη μνήμη για το ίδιο issue. Ο στόχος είναι να βρω ένα σύνολο τιμών που να είναι μεγαλύτερες από ένα δεδομένο Bid που δέχεται ο αλγόριθμος σαν είσοδο ενώ παράλληλα για

να επιταχυνθούν οι διαπραγματεύσεις κρατάω και σε μια μεταβλητή όλα τα relative utilities που έχω προσθέσει στο κάθε issue που χρησιμοποιούνται για να χαλαρώσουν με το πέρασμα του χρόνου τις προϋποθέσεις αποδοχής ενός νέου bid.

```

43: function RELATIVEUTILITYSEARCH(thresholdBid, issues, valueRelativeUtility)
    ▷ valueRelativeUtility contains own relative utility value matrix of each issue for each value in the utility space
44:   bid ← maxBid
45:   d ← thresholdBid.utility() − 1.0                                ▷ Difference from maximum utility value
46:   concessionSum ← 0
47:   randomIssues ← issues.shuffle()                                ▷ Must be in random order
48:   for issue in randomIssues do
49:     randomValues ← issue.getValues().shuffle()
50:     for value in randomValues do
51:       relativeUtility ← valueRelativeUtility.get(issue).get(value)
52:       if d ≤ concessionSum + relativeUtility then
53:         bid ← bid.putValue(issue, value)
54:         concessionSum += relativeUtility
55:       end if
56:     end for
57:   end for
58:   return bid
59: end function

```

1

**soft/hardConcessionThreshold:** Μέρος της στρατηγικής του πράκτορα με τα thresholds, χρησιμοποιούνται για τη επιτάχυνση των διαπραγματεύσεων στις τελευταίες στιγμές του γύρου. Το soft concession threshold χρησιμοποιείται μόνο όταν παίζει με μνήμη ο πράκτοράς μας και όταν ο κανονικοποιημένος χρόνος περάσει μια τιμή (πχ. 0.9) τότε αρχίζει να προσφέρει offers σε πράκτορες τα οποία είχαν δεχθεί ή είχαν κάνει στο παρελθόν με σκοπό να τελειώσουν γρηγορότερα οι διαπραγματεύσεις. Αντίστοιχα το hard concession threshold θα έχει μεγαλύτερη τιμή από το soft και όταν ο χρόνος περάσει και αυτό το κατώφλι θα αρχίσει ο πράκτοράς μας να προσφέρει τα καλύτερα offers που έχει κάνει ο κάθε πράκτορας κατά τη διάρκεια του γύρου, αυτή η τεχνική φαίνεται να λειτουργεί σχεδόν πάντα και για αυτό θα πρέπει να το συγκεκριμένο threshold να χρησιμοποιείται μόνο κατά το τέλος του γύρου διαφορετικά οι απολαβές του πράκτορα πιθανότατα να είναι πολύ μικρές.

**memoryDepth:** Η χρήση αυτής της παραμέτρου γίνεται μόνο όταν υπάρχει data persistency και καθορίζει για πόσους γύρους πριν θα κρατήσω στατιστικά στη μνήμη. Επειδή η μνήμη μας είναι περιορισμένη δεν μπορεί ο πράκτορας πάντα να κρατάει στη μνήμη του όλα τα στατιστικά για όλους τους γύρους που έχει παίξει. Τυπική τιμή της παραμέτρου είναι 4, μιας και πειραματικά τα αποτελέσματα δεν βελτιώνονται.

**cutOffValue:** Η τιμή κάτω από την οποία θεωρούμε ότι τα μεγέθη είναι μηδενικά. Επειδή αρκετά συχνά τα μεγέθη που εξετάζουμε παίρνουν πολύ μικρές τιμές (μικρότερες του  $10^{-6}$ ) χρειάζεται να ορίσουμε εμείς ποιο είναι το 0. Τυπικές τιμές είναι το  $10^{-4}$  και το  $10^{-5}$ .

**AgentStrategy:** Υποστηρίζονται 3 στρατηγικές από τον πράκτορά μας, η στρατηγική του TimeAgent , η στρατηγική του Threshold Agent και μια μίξη τους που κάνει τον πράκτορα να εναλλάσσει κάθε γύρο του τουρνουά στρατηγικές με βάση κάποιες παραμέτρους. Περισσότερα για την επιλογή αυτής της παραμέτρου μετά τα πειράματα.

**timeScalingFactor:** Αφορά μόνο τη στρατηγική του TimeAgent και χρησιμοποιείται για την κλιμάκωση της καμπύλης που φτιάχνει ο πράκτορας με βάση την οποία αποδέχεται ή απορρίπτει τα bids των αντιπάλων του. Η ιδέα είναι ότι σε περίπτωση που χρησιμοποιείται μνήμη ο πράκτορας μπορεί να δει αν ο αντίπαλός του χρειάστηκε αρκετούς γύρους για να καταλήξει σε κάποια συμφωνία παλιότερα και να “χαμηλώσει” την καμπύλη του ελαφρώς για να μην σπαταλήσει ξανά γύρους μέχρι να φτάσει σε κάποιο σημείο συμφωνίας. Αυτή η παράμετρος αν και έχει πολύ περιορισμένη χρήση ευνοεί συχνά τον πράκτορά μας όταν υπάρχει discount και ο αντίπαλος έχει στρατηγική Boulware.

**SimulatedAnnealingParams:** Όπως φαίνεται και από το όνομα αυτή η παράμετρος καθορίζει τη συμπεριφορά του αλγορίθμου Simulated Annealing που χρησιμοποιεί ο πράκτοράς μας. Η τιμές που έχει αυτή η δομή είναι η αρχική, η τελική θερμοκρασία, ο παράγοντας ψύξης και ο αριθμός επαναλήψεων για την επιλογή της τυχαίας τιμής (default: 1) , αν δηλαδή είναι μεγαλύτερη του 1 τότε αρχικά αντί να επιλέξει μια τυχαία τιμή ο αλγόριθμος θα επιλέξει παραπάνω και θα κρατήσει την καλύτερη από αυτές, κάτι τέτοιο φυσικά καθυστερεί αρκετά τον αλγόριθμο.

**vetoVal , bidUtilThreshold:** Αυτές οι παράμετροι αν και στην πράξη δεν χρησιμοποιήθηκαν αξίζει να αναφερθούν. Ο σκοπός της πρώτης ήταν να θέσει ένα κατώτερο όριο στην τιμή των utilities των offers που αποδέχεται ο πράκτοράς μας ενώ η 2η θεωρούσε ότι το μεγαλύτερο utility ενός bid μπορεί να φτάσει μέχρι την τιμή της, από εκεί και μετά όλα τα bids έχουν την ίδια τιμή, κάτι που έκανε τον πράκτορα να ξεκινήσει τις διαπραγματεύσεις κάνοντας χαμηλότερες προσφορές. Στην πράξη και οι 2 παράμετροι παρέκαμπταν ένα μεγάλο μέρος της στρατηγικής του πράκτορα με αποτέλεσμα να μην κυλάνε ομαλά οι διαπραγματεύσεις (πχ. απότομα ανεβοκατεβάσματα σε utility offers) και για αυτό στο τέλος εγκαταλείφθηκαν.

## Πειραματικά αποτελέσματα

Σε αυτή την ενότητα θα τρέξουμε στην πλατφόρμα GENIUS 7.1.6 τόσο σε tournament όσο και σε negotiation mode διάφορα πειράματα για τις παρακάτω υλοποιήσεις.

1. Threshold Agent with Standard Data Persistency
2. Threshold Agent with Disabled Data Persistency
3. Time Agent with with Standard Data Persistency
4. Time Agent with Disabled Data Persistency

Οι παραπάνω πράκτορες θα έχουν συχνά διαφορετικές παραμέτρους αλλά σε περίπτωση που δεν αναγράφεται οι τιμές θα έχουν τις ακόλουθες τιμές:

**searchingMethod:** Simmulated Annealing

**SimmulatedAnnealingParams:**

startTemp: 1.000  
endTemp: 0.001  
coolingFactor: 0.999  
Step (αριθμός επιλογής αρχικών τιμών): 1

**TimeSaclingFactor:** 1.0

**CutOffValue:**  $10^{-6}$

**SoftConcessionThreshold:** 0.70

**HardConcessionThreshold:** 0.98

**MemoryDepth:** 1

**Data Persistency:** Standard

Κατά τη διάρκεια των περισσότερων πειραμάτων οι αντίπαλοι θα είναι συνήθως ο Atlas32016 και ο Caduceus. Ο 1ος είναι ένας από τους πράκτορες που έπρεπε να κερδίσουν οι διαγωνιζόμενοι το 2016 για να κερδίσουν τα προκριματικά ενώ ο 2ος ήταν ο νικητής του περσινού διαγωνισμού. Για κάθε πράκτορα θα τρέχουμε αρχικά διάφορα negotiations μέχρι να δούμε ποιες παραμέτρους πρέπει να κρατήσουμε ή να αλλάξουμε, στη συνέχεια θα βάζουμε τον πράκτορα σε τουρνουά 180 γύρων και θα βλέπουμε αν βελτιώθηκε ή πρέπει να αλλάξουμε κάποιες παραμέτρους. Επίσης ο πράκτορας θα παίζει σε τουρνουά 60 ή 120 sec ανά γύρο για να πάρουμε τις τελικές μετρήσεις, συνήθως το avg utility του πράκτορα και το social welfare. Τέλος τα πειράματα εκτελέστηκαν σε υπολογιστή με 16GB RAM και 4 CPU @ 3.2GHz.

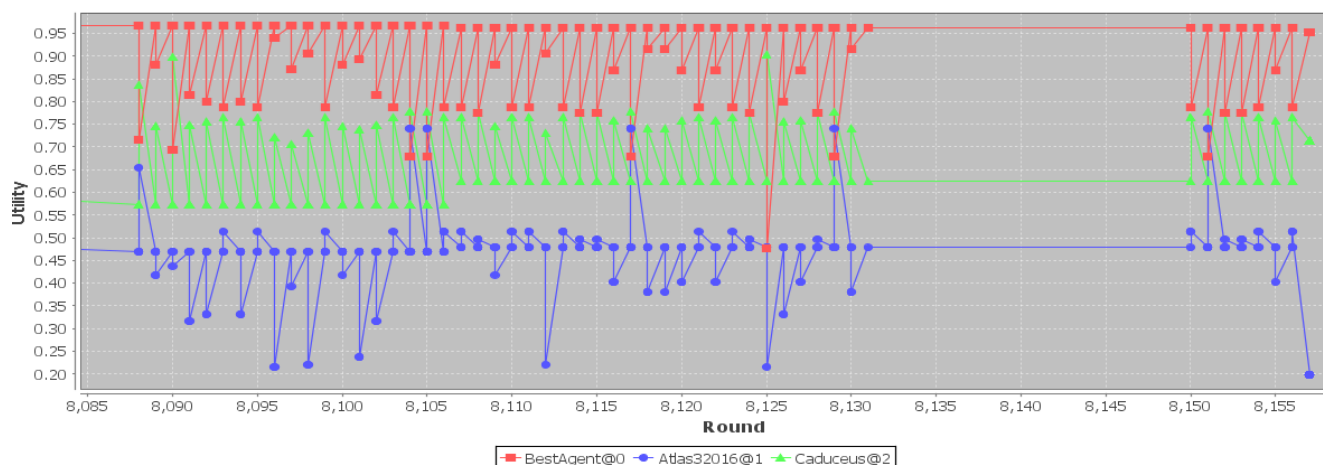
## Negotiations των πρακτόρων στα 180 sec με default τιμές

Negotiations των πρακτόρων μου με default παραμέτρους (οι αντίπαλοι έχουν πάντα μνήμη) για 180 sec, τα χρώματα είναι ίδια (ο δικός μου είναι πάντα ο 1ος με κόκκινο χρώμα) για όλα τα ακόλουθα negotiations. Σε περίπτωση που υπάρχουν στα κενά στα σχήματα (bug του

GENIUS) απλά θεωρήστε ότι η απόσταση είναι ενός γύρου μόνο και όχι όσων αναγράφονται, αυτό συμβαίνει λόγω φόρτου στη CPU. Τέλος στις εικόνες συνήθως φαίνονται οι τελευταίοι ~100 γύροι για να είναι πιο ξεκάθαρα τα σχήματα.

### Πράκτορας 1 ( Threshold Agent with Standard Data Persistency)

Time (s): 176.457772295  
 Rounds: 8157  
 Agreement?: Yes  
 Discounted?: No  
 3  
 Min. utility: 0.19881  
 Max. utility: 0.95250  
 Distance to pareto: 0.00000  
 Distance to Nash: 0.74875  
 Social welfare: 1.86396  
 Agent utility: 0.95250 (BestAgent@0) (RED)  
 Agent utility: 0.19881 (Atlas32016@1) (BLUE)  
 Agent utility: 0.71265 (Caduceus@2) (GREEN)

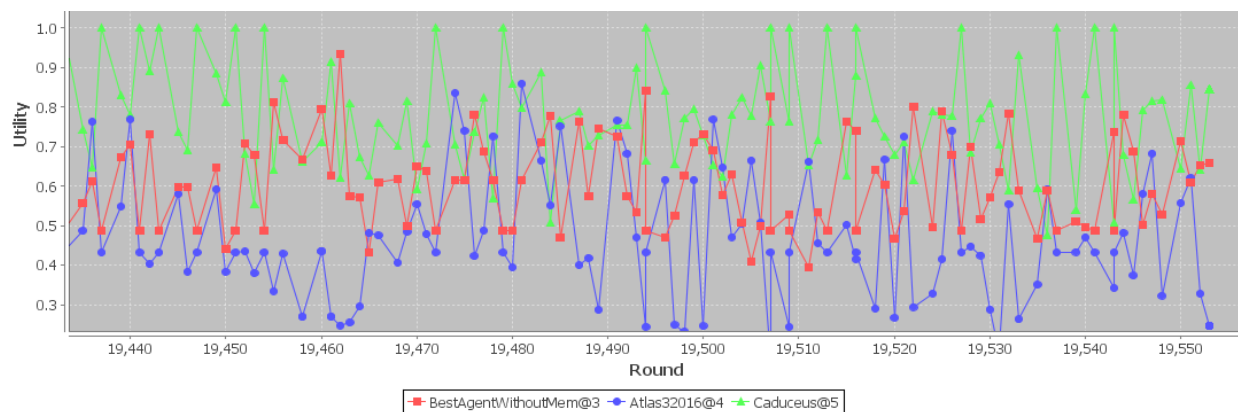


### Πράκτορας 2 ( Threshold Agent with Data Persistency Disabled):

Time (s): 149.461964602  
 Rounds: 19553  
 Agreement?: Yes  
 Discounted?: No  
 3  
 Min. utility: 0.24522  
 Max. utility: 0.84548  
 Distance to pareto: 0.08789  
 Distance to Nash: 0.70665  
 Social welfare: 1.74903  
 Agent utility: 0.65833 (BestAgentWithoutMem@3)

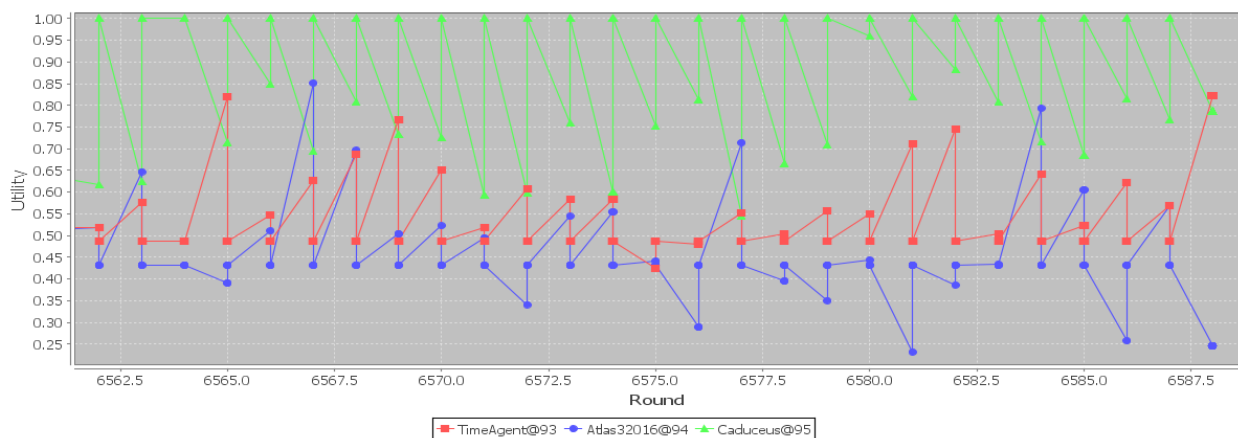


Agent utility: 0.24522 (Atlas32016@4)  
 Agent utility: 0.84548 (Caduceus@5)



### Πράκτορας 3 ( Time Agent with Standard Data Persistency)

Time (s): 149.458837572  
 Rounds: 6588  
 Agreement?: Yes  
 Discounted?: No  
 3  
 Min. utility: 0.24638  
 Max. utility: 0.82250  
 Distance to pareto: 0.07734  
 Distance to Nash: 0.68668  
 Social welfare: 1.85534  
 Agent utility: 0.82250 (TimeAgent@93)  
 Agent utility: 0.24638 (Atlas32016@94)  
 Agent utility: 0.78647 (Caduceus@95)

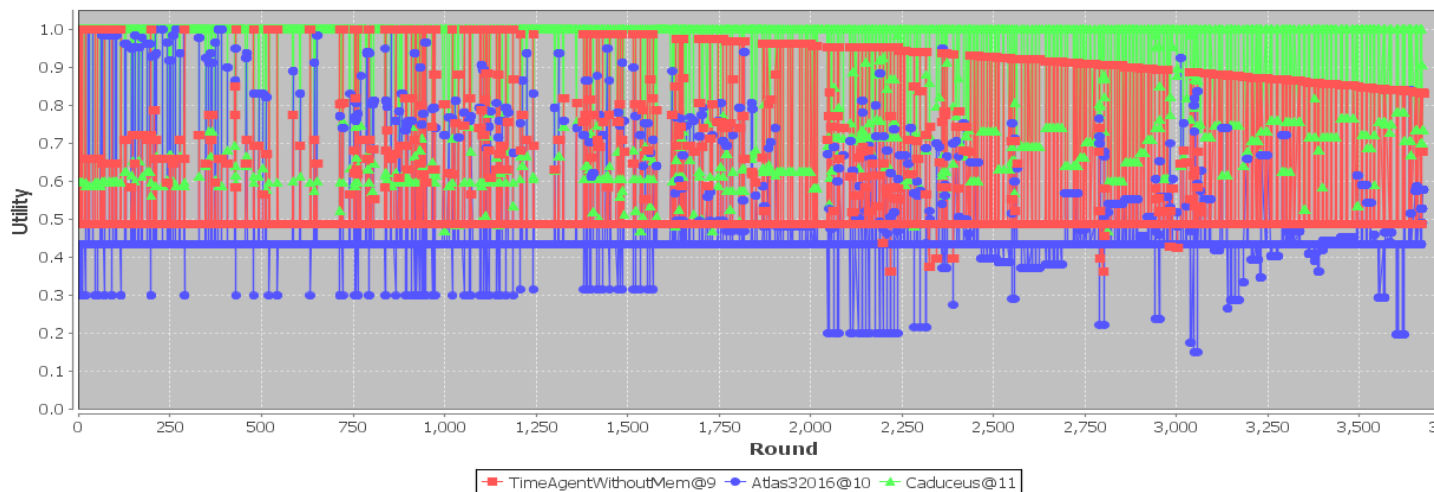


### Πράκτορας 4 ( Time Agent with Data Persistency Disabled):

Time (s): 149.697366089  
 Rounds: 3674  
 Agreement?: Yes  
 Discounted?: No

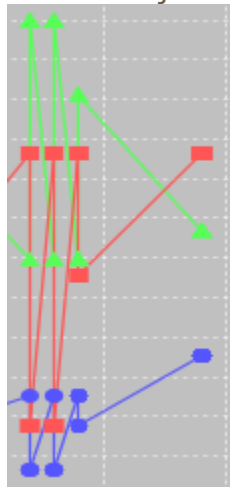
3

Min. utility: 0.57576  
 Max. utility: 0.83083  
 Distance to pareto: 0.04996  
 Distance to Nash: 0.35615  
 Social welfare: 2.14080  
 Agent utility: 0.83083 (TimeAgentWithoutMem@9)  
 Agent utility: 0.57576 (Atlas32016@10)



Agent utility: 0.73420 (Caduceus@11)

Zoom στους τελευταίους 5 γύρους:



### Αποτελέσματα - Σχόλια (NM -> Χωρίς μνήμη)

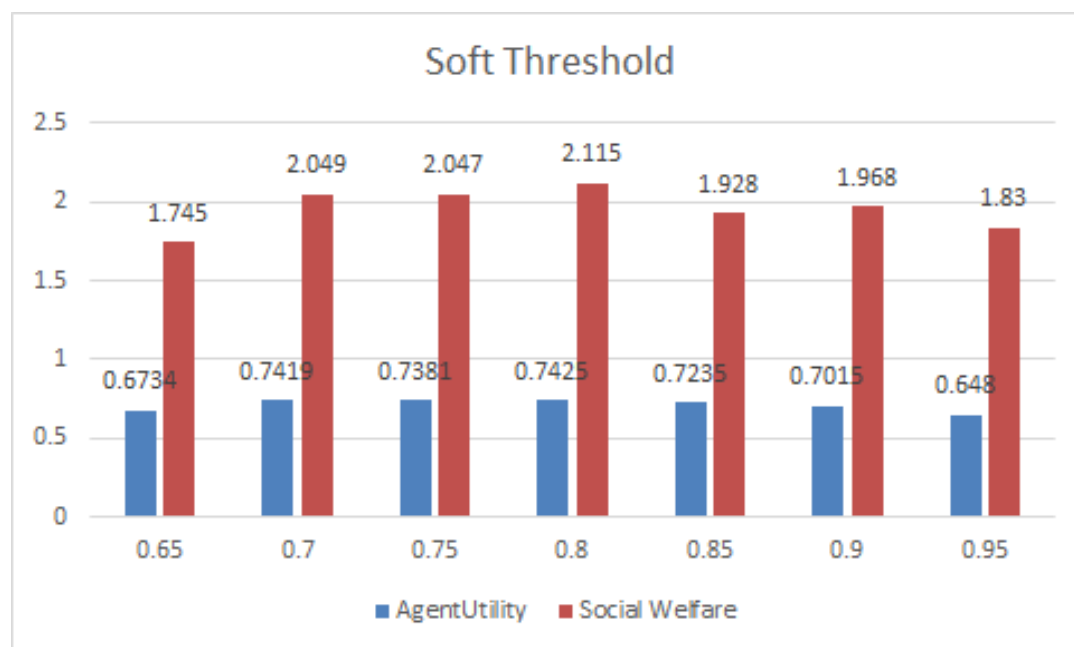
	1.Threshold	2.Threshold NM	3.TimeAgent	4.TimeAgent NM
Utility	0.95	0.658	0.822	0.830
Social Welfare	1.86	1.74	1.85	2.14
Total Rounds	8157	19553	6588	3674
Time	176.45	149.46	149.45	149.67

Μέχρι στιγμής τα αποτελέσματα δείχνουν ότι ο 1ος πράκτορας πετυχαίνει το μεγαλύτερο utility αλλά ο 4ος πετυχαίνει το καλύτερο social welfare κάτι που πιθανόν να μας ενδιαφέρει, ειδικά σε αποτελέσματα τουρνουά που θα δούμε στη συνέχεια. Επίσης αξίζει να σημειωθεί ότι μόνο ο 1ος πράκτορας χρησιμοποίησε σχεδόν όλο το χρόνο που μπορούσε (176.45 sec / 180.00 sec) ενώ στην εικόνα του πράκτορα 4 (χωρίς zoom) μπορεί κανείς να δει την κόκκινη καμπύλη που σχηματίζεται εξαιτίας της στρατηγικής που ακολουθεί.

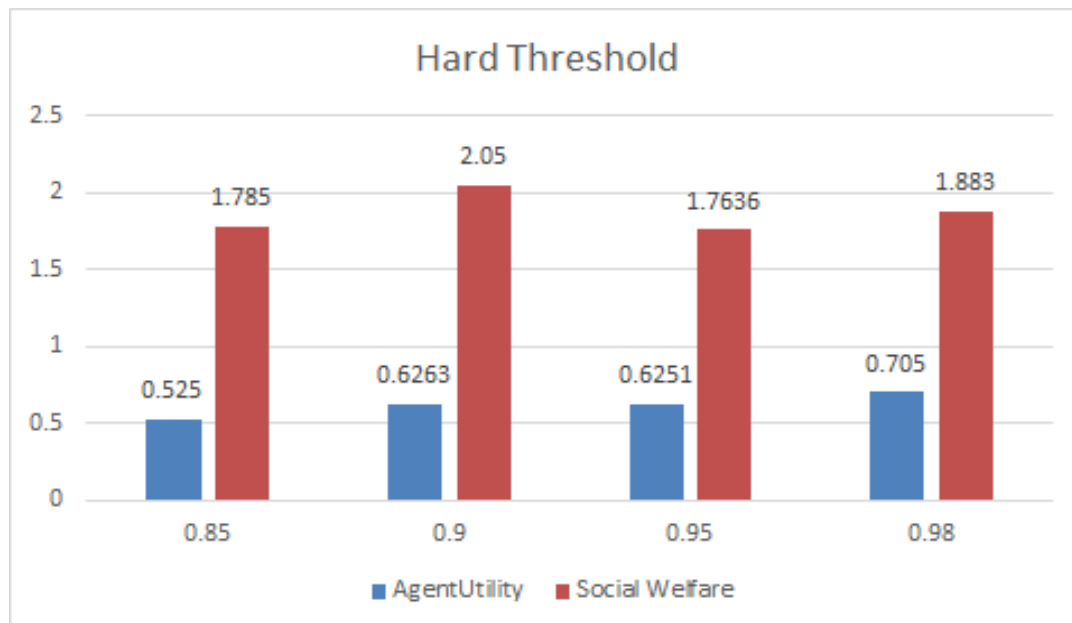
### Βελτιστοποίηση παραμέτρων σε τουρνουά

Σε αυτή την υπο-ενότητα θα προσπαθήσουμε να βελτιστοποιήσουμε τις παραμέτρους για τις διάφορες υλοποιήσεις του πράκτορά μας. Αρχικά θα εστιάσουμε στον 1ο πράκτορα (Threshold Agent with Standard Data Persistency) που φαίνεται να μεγιστοποιεί το utility του και στη συνέχεια στον Time Agent.

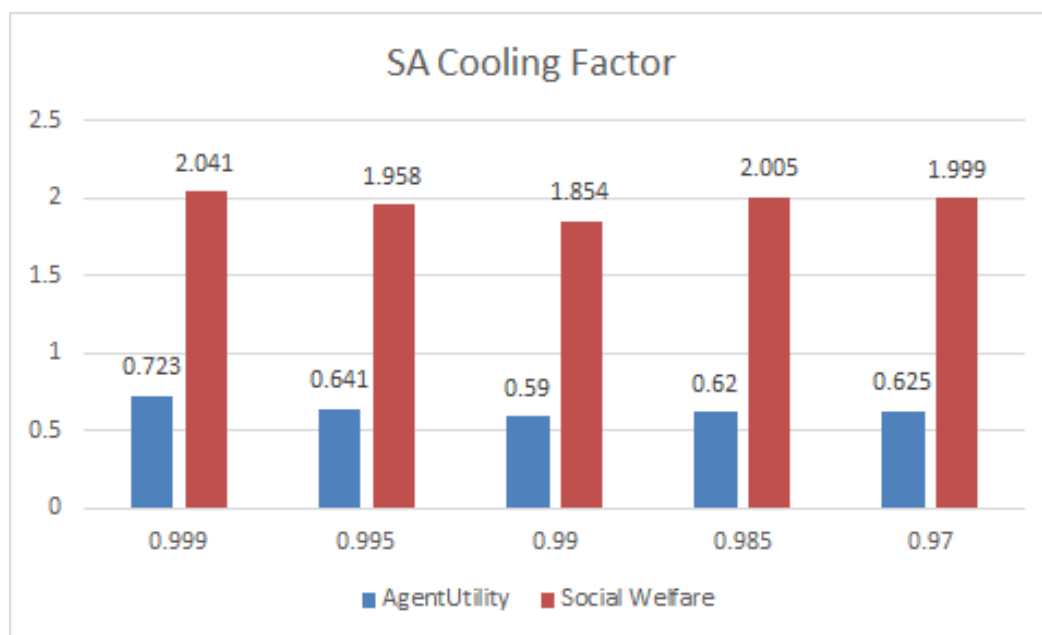
Tournament Results για διάφορες παραμέτρους στον 1ο πράκτορα.



Άρα επιλέγω ως βέλτιστη τιμή το 0.8 επειδή μεγιστοποιεί τόσο το SW όσο και το utility του πράκτορα, μια αναμενόμενη τιμή μιας και περιμέναμε ότι οι φιλικές τιμές προσφοράς προς τους αντιπάλους θα επιτάχυναν τις διαπραγματεύσεις για αυτό και δεν θα έπρεπε να γίνεται από νωρίς.

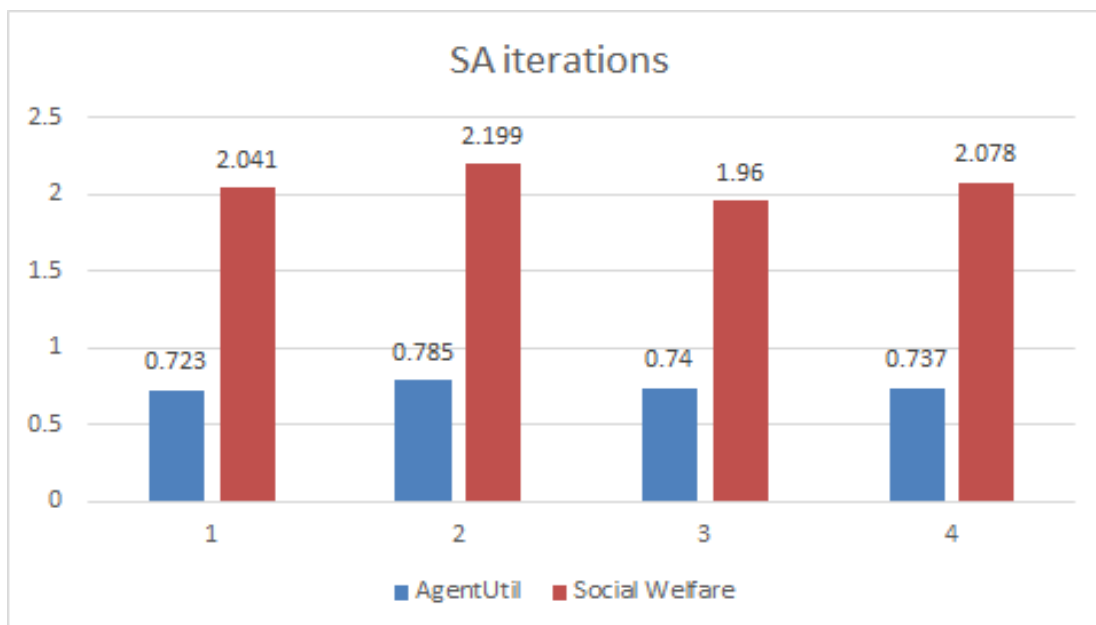


Από ότι βλέπουμε η τιμή που μεγιστοποιεί το SW είναι για 0.90 αλλά η τιμή που μεγιστοποιεί το utility του πράκτορα είναι η 0.98, η οποία και θα επιλεγεί τελικά. Ο λόγος που δεν επιλέγουμε >0.99 είναι επειδή σε μικρό αριθμό γύρων δεν θα προλάβει να ενεργοποιηθεί καθόλου.

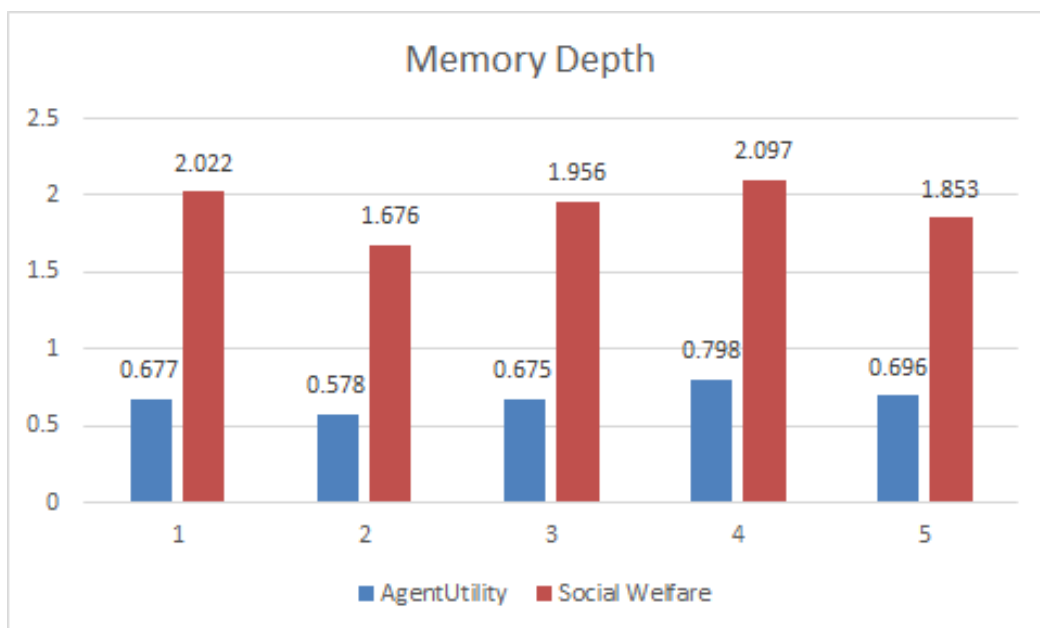


Το παραπάνω γράφημα δείχνει τα αποτελέσματα για διάφορες τιμές της παραμέτρου cooling factor στο Simmulated Annealing που γίνεται. Μικρές τιμές προσφέρουν πιο ακριβή αλλά πιο

αργά αποτελέσματα, πειραματικά βλέπουμε ότι για 0.999 μεγιστοποιούνται τα utility και SW ενώ για τιμές  $>0.999$  και ο αλγόριθμος καθυστερεί υπερβολικά πολύ για αυτό το usecase.

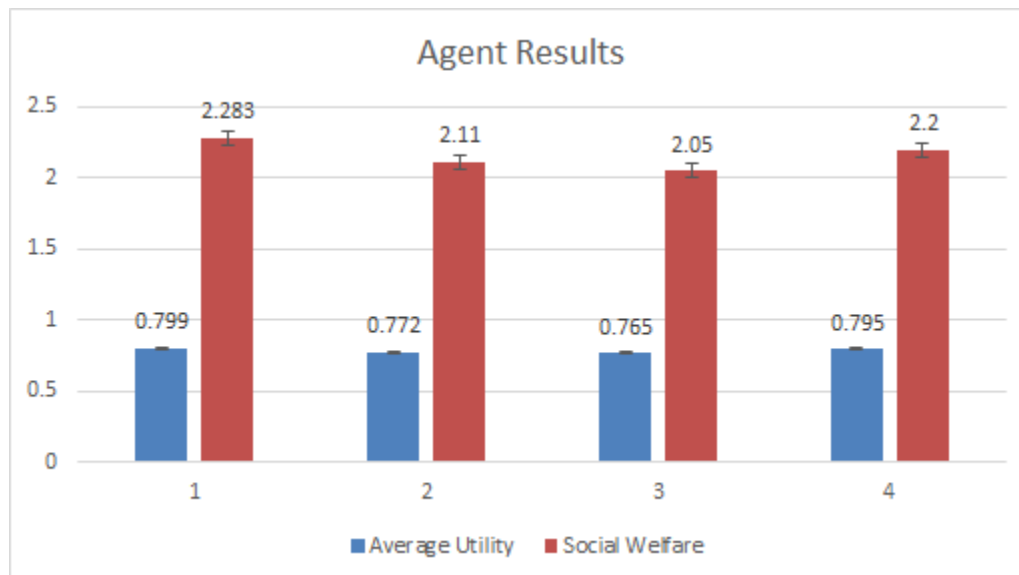


Στο προηγούμενο σχήμα βλέπουμε τα αποτελέσματα του SA για διαφορετικό αριθμός επιλογής αρχικών τιμών. Όσο μεγαλύτερη η τιμή τόσο πιο αργός (αλλά και ακριβής) ο αλγόριθμος και όπως βλέπουμε η τιμή 2 είναι η βέλτιστη.



Τέλος βλέπουμε στο παραπάνω σχήμα την επίδραση στο μέγεθος της μνήμης που μπορούμε να κρατήσουμε. Δοκιμάζουμε την απόδοση της μνήμης του πράκτορα όταν κρατάει στατιστικά για 1-5 γύρους τουρνουά στο παρελθόν. Παρατηρούμε ότι για βάθος ίσο με 4 βελτιστοποιείται η απόδοση του πράκτορά μας.

Αποτελέσματα tournament mode για χρόνο γύρου 60sec.



Τα αποτελέσματα είναι εμφανή, ο 1ος πράκτορας (Threshold Agent with memory) είναι ο καλύτερος και αυτός θα είναι η επιλογή μου για τον ANAC. Από εδώ και κάτω θα ασχοληθούμε με τη βελτιστοποίηση του. Να σημειωθεί επίσης ότι και ο Time Agent ήρθε 2ος με πολύ μικρή διαφορά με αποτέλεσμα να γίνει μελέτη και ενός πράκτορα που αλλάζει στρατηγική ανάμεσα σε αυτές τις 2.

Ακολουθούν τα αποτελέσματα από ένα negotiation με άλλους 4 πράκτορες:

Time (s): 171.056618928

Rounds: 3004

Agreement?: Yes

Discounted?: No

5

Min. utility: 0.35555

Max. utility: 0.84833

Distance to pareto: 0.00000

Distance to Nash: 0.32219

Social welfare: 3.52561

Agent utility: 0.84833 (BestAgent@0) ←---- Δικός μου

Agent utility: 0.35555 (Atlas32016@1)

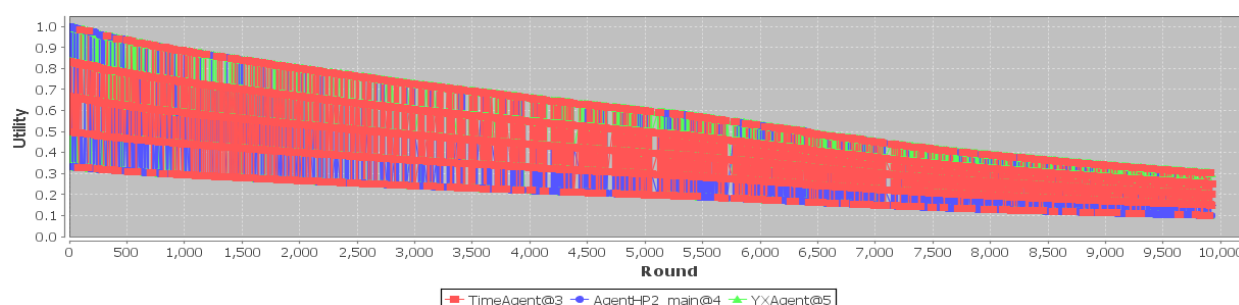
Agent utility: 0.78829 (Caduceus@2) ←---- Περσινή 1η θέση

Agent utility: 0.83333 (ParsAgent2@3)

Agent utility: 0.70010 (YXAgent@4) ←---- Περσινή 2η θέση

Ακολουθεί ένα negotiation με issues από το domain Farma πρώτα από τον TimeAgent και μετά από τον πράκτορα που επιλέξαμε για τον ANAC (discounted):

Time (s): 93.947684411  
 Rounds: 9919  
 Agreement?: Yes  
 Discounted?: Yes  
 3  
 Min. utility: 0.15033  
 Max. utility: 0.25055  
 Distance to pareto: 0.00000  
 Distance to Nash: 0.21842  
 Social welfare: 0.65142  
 Agent utility: 0.25055 (TimeAgent@3)  
 Agent utility: 0.15033 (AgentHP2\_main@4)  
 Agent utility: 0.25055 (YXAgent@5)

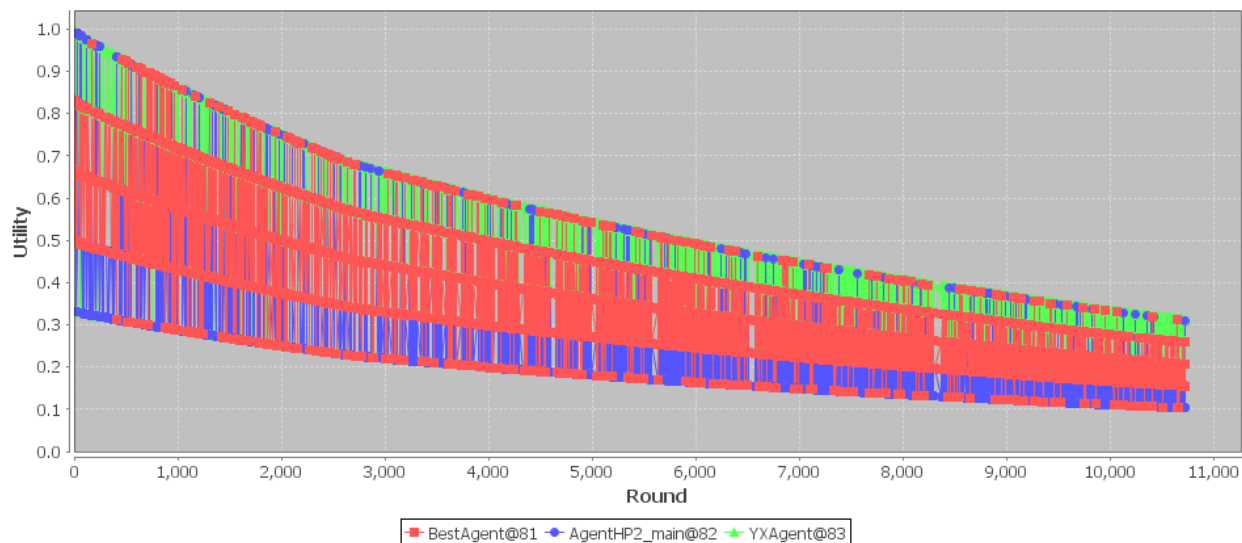


Εδώ βλέπουμε ότι αυτός ο πράκτορας αν και δεν κατάφερε να κερδίσει τον YXAgent έφερε ισοπαλία, τα παραπάνω utilities είναι discounted για αυτό και η φθίνουσα καμπύλη.

Θα ήθελα να τελειώσω αυτή την αναφορά με τα αποτελέσματα του παραπάνω negotiation αλλά για τον πράκτορα που επιλέχθηκε για τον ANAC.

Time (s): 91.557582714  
 Rounds: 10727  
 Agreement?: Yes  
 Discounted?: Yes  
 3  
 Min. utility: 0.15499  
 Max. utility: 0.25832  
 Distance to pareto: 0.00000  
 Distance to Nash: 0.08949  
 Social welfare: 0.67164  
 Agent utility: 0.25832 (BestAgent@81)  
 Agent utility: 0.15499 (AgentHP2\_main@82)  
 Agent utility: 0.25832 (YXAgent@83)





Παρατηρούμε ότι η καμπύλη φαίνεται παρόμοια αλλά τα αποτελέσματα είναι καλύτερα τόσο σε avg utility του πράκτορά μας όσο και σε social welfare..

## Βιβλιογραφία - Πηγές

- Multi-agent and Complex Systems  
Editors: Bai, Q., Ren, F., Fujita, K., Zhang, M., Ito, T. (Eds.)
- Διαφάνειες του μαθήματος ΠΛΗ517.
- Σελίδα του Genius: <http://ii.tudelft.nl/genius/>
- Η βοηθός του μαθήματος.