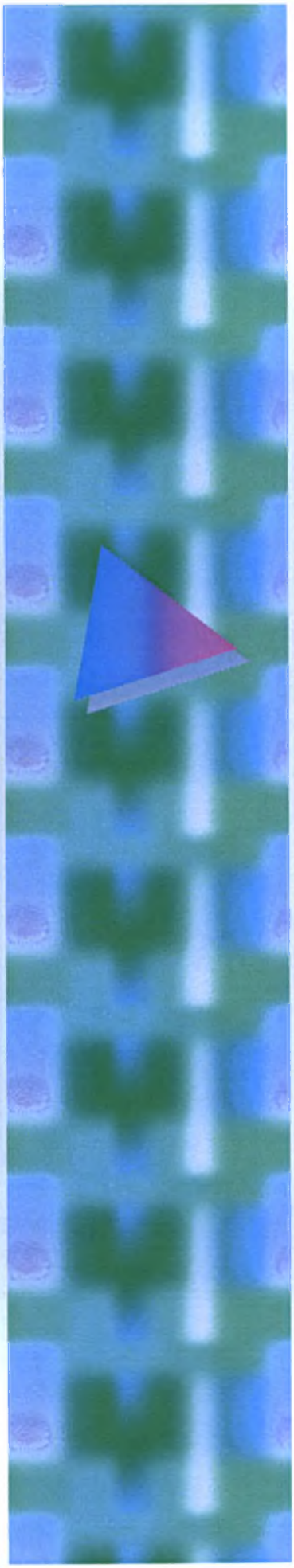


Πανεπιστήμιο Θεσσαλίας
Πολυτεχνική Σχολή
Τμήμα Μηχανικών Η/Υ,
Τηλεπικοινωνιών και Δικτύων

Μεταπτυχιακή Εργασία

Συνεργατική Κατανεμημένη Επίλυση Προβλήματος με έμφαση στην Αυτόματη Διαπραγμάτευση μεταξύ Πρακτόρων

Δήμητρα Σαρδέλλη

Επιβλέπουσα
Ασπασία Δασκαλοπούλου

Μάιος 2007

Περιεχόμενα.

Εισαγωγή.

Τα πολυπρακτορικά συστήματα αποκτούν αυξανόμενη σημασία καθώς διευρύνεται η χρήση των τεχνολογιών πληροφορικής και επικοινωνιών σε περισσότερο πολύπλοκους τομείς και είναι πλέον επιθυμητή η σύνθεση πληροφοριών, υπολογισμού και υπηρεσιών που παρέχονται σε/από αντίστοιχα πλέγματα. Τα ανοιχτά πολυπρακτορικά συστήματα κατοικούνται από κινητούς πράκτορες των οποίων οι ιδιότητες δεν είναι εκ των προτέρων γνωστές. Έτσι το περιβάλλον με το οποίο αλληλεπιδρά ένας πράκτορας σε τέτοιο σύστημα είναι μη-ντετερμινιστικό, απρόσιτο και δυναμικό. Έχει αναπτυχθεί πολλή θεωρία και κάποια υπολογιστικά εργαλεία για να ρυθμιστούν οι αλληλεπιδράσεις πρακτόρων σε τέτοια περιβάλλοντα, ορίζοντας νόρμες για το τι επιτρέπεται/απαγορεύεται/υποχρεούται να κάνει κάθε πράκτορας και μηχανισμούς με τους οποίους οι πράκτορες διαπραγματεύονται αλλαγές στις νόρμες της κοινωνίας τους. Στην παρούσα εργασία σχεδιάστηκε, υλοποιήθηκε και μελετήθηκε ένα απλό σύστημα διαπραγμάτευσης μεταξύ δύο πρακτόρων μέσω του οποίου πραγματοποιήθηκε σύγκριση μεταξύ διαφορετικών πρωτοκόλλων, πολιτικών και μεθόδων που χρησιμοποιούν οι πράκτορες.

Ειδικότερα, στο κεφάλαιο 1 παρέχεται το βασικό υπόβαθρο γνώσεων απαραίτητο για τη συνέχεια. Ορίζονται οι έννοιες της συνεργατικής κατανεμημένης επίλυσης προβλήματος, της διαπραγμάτευσης μεταξύ πρακτόρων, των τομέων προσανατολισμένων σε εργασίες και του μονοτονικού πρωτοκόλλου συμβιβασμού. Στο κεφάλαιο 2 μέσα από θεωρητική μελέτη αναλύεται λεπτομερέστερα η λειτουργία του μονοτονικού πρωτοκόλλου, συζητούνται εναλλακτικές λύσεις και γίνεται απόπειρα υπολογιστικών βελτιώσεων. Στο κεφάλαιο 3 αναλύεται, σχεδιάζεται και υλοποιείται ένα απλό σύστημα διαπραγμάτευσης δύο πρακτόρων. Στο κεφάλαιο 4 πραγραγματοποιείται σύγκριση των χρησιμοποιούμενων μεθόδων και πολιτικών των πρακτόρων μέσω πειραματισμού με διάφορα σενάρια χρήσης. Με τον τρόπο αυτό επαληθεύονται πρακτικά τα θεωρητικά συμπεράσματα του κεφαλαίου 2. Στο κεφάλαιο 5 γίνεται απόπειρα σύγκρισης του μονοτονικού πρωτοκόλλου συμβιβασμού με ένα εναλλακτικό πρωτόκολλο. Τέλος, στο κεφάλαιο 6 πραγματοποιείται αναφορά σε κάποια ανοιχτά ζητήματα που υπάρχουν στο χώρο αυτό και που θα μπορούσαν να αποτελέσουν βελτιώσεις της παρούσας αλλά και αντικείμενο μελοντικής εργασίας.

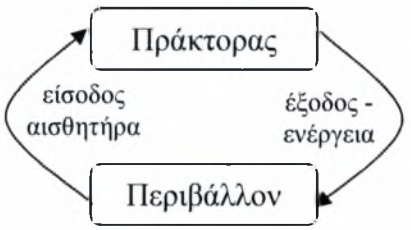
1. Βασικό υπόβαθρο.

Η *συνεργατική κατανεμημένη επίλυση προβλήματος*[1] – ΣΚΕΠ (cooperative distributed problem solving) μελετά πως μια ομάδα αυτόνομων πρακτόρων επιλέγουν να συνεργαστούν για την επίτευξη ενός κοινού στόχου, για να επιλύσουν προβλήματα που ξεπερνούν τις ατομικές τους ικανότητες και απαιτούν συλλογική προσπάθεια.

1.1. Πράκτορες και πολυπρακτορικά συστήματα.

Πράκτορας (agent) είναι ένα υπολογιστικό σύστημα που λειτουργεί σε ένα περιβάλλον και είναι ικανό για ανεξάρτητη δράση για λογαριασμό του χρήστη / ιδιοκτήτη του, δηλαδή μπορεί να αποφασίσει μόνο του τι πρέπει να κάνει για να ικανοποιήσει τους στόχους για τους οποίους σχεδιάστηκε, χωρίς να χρειάζεται εντολές ανά πάσα στιγμή.[2]

Το σχήμα 1.1 παρουσιάζει αφηρημένα την αλληλεπίδραση ενός πράκτορα με το περιβάλλον του. Ο πράκτορας αντιλαμβάνεται μέσω των αισθητήρων του την κατάσταση του περιβάλλοντος, επεξεργάζεται αυτήν την πληροφορία και αποφασίζει ποια ενέργεια θα πρέπει να εκτελέσει μέσω των μηχανισμών δράσης του.



Σχήμα 1.1

Υπάρχει μεγάλη συζήτηση σχετικά με το ποια χαρακτηριστικά θα πρέπει να διαθέτει ένα πρόγραμμα λογισμικού προκειμένου να χαρακτηρίζεται πράκτορας. Οι περισσότεροι ερευνητές συμφωνούν ότι τα παρακάτω τρία είναι τα ελάχιστα απαιτούμενα χαρακτηριστικά:

- *Αντιδραστικότητα / ορθολογισμός* (reactivity / rationality): οι νοήμονες πράκτορες μπορούν να αντιλαμβάνονται το περιβάλλον τους και να απαντούν εντός λογικού χρόνου σε αλλαγές που συμβαίνουν σε αυτό, με σκοπό να εκπληρώσουν τους στόχους τους.

[1] [Durfee 1999 κεφ.3], [Wooldridge 2002 κεφ.9]
[2] [Wooldridge 2002 κεφ.1 & 2]

- *Ενεργητικότητα / αυτονομία* (proactiveness / autonomy): οι νοήμονες πράκτορες μπορούν να πάρουν πρωτοβουλίες και να ενεργήσουν προς την εκπλήρωση των στόχων τους.
- *Κοινωνική ικανότητα* (social ability): οι νοήμονες πράκτορες μπορούν να αλληλεπιδρούν με άλλους πράκτορες (και ανθρώπους) για να εκπληρώσουν τους στόχους τους. Αυτή η αλληλεπίδραση δεν περιορίζεται μόνο στην ανταλλαγή δεδομένων αλλά έχει χαρακτηριστικά που την κάνουν να μοιάζει με ανθρώπινη αλληλεπίδραση (δηλαδή διαπραγματεύονται, επιχειρηματολογούν, υπόσχονται, συνάπτουν συμφωνίες κλπ)

Κατά αντιστοιχία με τις ανθρώπινες κοινωνίες, κάθε πράκτορας είναι ικανός για ανεξάρτητη εργασία. Επειδή όμως διαθέτει ελλιπή πείρα, πόρους ή πληροφορίες, ίσως ορισμένα προβλήματα να μην μπορεί να τα επιλύσει καθόλου μόνος του, ή κάποια άλλα να μην μπορεί να τα επιλύσει όσο γρήγορα και αποτελεσματικά απαιτείται. Η λύση είναι η συνεργασία, έτσι ώστε διαφορετικοί πράκτορες να επιλύσουν διαφορετικά τμήματα του συνολικού προβλήματος και στη συνέχεια οι επιμέρους λύσεις να συνδυαστούν στη συνολική λύση.

Πολυπρακτορικό σύστημα[3] - ΠΣ (multiagent system) είναι μια κοινωνία από πράκτορες που αλληλεπιδρούν μεταξύ τους, ανταλλάσσοντας μηνύματα μέσω μιας δικτυακής υποδομής. Στην γενική περίπτωση κάθε πράκτορας εκπροσωπεί χρήστες ή φορείς με πολύ διαφορετικά συμφέροντα και κίνητρα. Για να αλληλεπιδράσουν επιτυχώς μεταξύ τους οι πράκτορες πρέπει να συνεργαστούν, να συντονιστούν και να διαπραγματευτούν, όπως και οι άνθρωποι στις δικές τους κοινωνίες.

Οι πράκτορες, ανάλογα με το κατά πόσο είναι πρόθυμοι για συνεργασία, διακρίνονται σε *φιλάνθρωπους* (benevolent) και *εγωιστές* (self-interested)[4]. Στην πρώτη περίπτωση, το σύστημα μοιράζεται ένα κοινό στόχο, που μπαίνει πάνω από τους επιμέρους στόχους και προτιμήσεις κάθε πράκτορα. Αυτό είναι για παράδειγμα εφικτό όταν οι πράκτορες έχουν σχεδιαστεί ή ανήκουν στον ίδιο φορέα. Αντίθετα, οι εγωιστές πράκτορες που αποτελούν και την πιο γενική περίπτωση, δεν μπορεί να υποτεθεί ότι μοιράζονται κοινό στόχο. Συχνά έχουν σχεδιαστεί από διαφορετικούς φορείς κι εκπροσωπούν διαφορετικά συμφέροντα. Για το λόγο αυτό συχνά προκύπτουν συγκρούσεις μεταξύ τους, ακριβώς όπως και στις ανθρώπινες κοινωνίες. Παρόλ' αυτά, αναγνωρίζουν την ανάγκη για συνεργασία, πάλι όπως και στις ανθρώπινες κοινωνίες.

---

[3] [Wooldridge 2002 κεφ.1]
[4] [Wooldridge 2002 κεφ.9]

1.2. Στάδια στη συνεργατική κατανεμημένη επίλυση προβλήματος.

Στη ΣΚΕΠ[5] σημαντικό ρόλο παίζει η *συνάφεια* (coherence) της ομάδας, που εκφράζει την επιθυμία για συνεργασία και το πόσο καλά συμπεριφέρονται σαν μια μονάδα. Επίσης, σημαντικός είναι ο *συντονισμός* (coordination) μεταξύ των πρακτόρων, δηλαδή ο βαθμός στον οποίο μπορούν να αποφευχθούν δραστηριότητες όπως συγχρονισμός και ευθυγράμμιση μεταξύ των πρακτόρων. Ακόμη, σπουδαία είναι η *ικανότητα* (competence) συνεργασίας, η γνώση για το πώς μπορούν να συνεργαστούν ικανοποιητικά.

Η ΣΚΕΠ περιλαμβάνει τέσσερα στάδια:

α) *Αναγνώριση* (recognition) από κάποιο πράκτορα της ανάγκης και δυνατότητας για συνεργασία.

β) *Δημιουργία ομάδας* (team formation) που τα μέλη της έχουν δώσει υπόσχεση για συνεργασία, μετά από πρωτοβουλία πράκτορα στο βήμα α.

γ) *Δημιουργία πλάνου* (plan formation) για την επίτευξη του επιθυμητού στόχου μετά από διαπραγμάτευση των πρακτόρων της ομάδας στο βήμα β.

δ) *Ομαδική εκτέλεση* (team action) του κοινά συμφωνημένου πλάνου.

Η επίλυση ενός μεγάλου προβλήματος στη ΣΚΕΠ περιλαμβάνει τα στάδια:

- *Αποσύνθεση προβλήματος* (problem / task decomposition) σε μια σειρά υποπροβλημάτων. Ζητήματα είναι το μέγεθος των υποπροβλημάτων και ο τρόπος επιτέλεσης της αποσύνθεσης.
- *Ανάθεση υποπροβλημάτων* (sub-problem / sub-task allocation) στους πράκτορες. Σε περίπτωση που οι πράκτορες έχουν τα ίδια χαρακτηριστικά και οι εργασίες είναι όμοιες μεταξύ τους, το στάδιο αυτό είναι τετριμένο. Διαφορετικά μπορεί να είναι πολύπλοκη διαδικασία, που πραγματοποιείται από κάποιον τρίτο ή μέσω διαπραγμάτευσης μεταξύ των πρακτόρων, όπως θα δούμε παρακάτω.
- *Επίλυση υποπροβλημάτων* (sub-problem / sub-task solution). Μπορεί να χρειαστεί και ανταλλαγή πληροφοριών ή αποτελεσμάτων (result sharing) μεταξύ των πρακτόρων. Ο πράκτορας που μοιράζεται αποτελέσματα κερδίζει σε ακρίβεια, ταχύτητα, αυτοπεποίθηση.
- *Σύνθεση λύσης* (solution synthesis) από τις επιμέρους λύσεις των υποπροβλημάτων. Όταν ένας πράκτορας επιλύει ένα υποπρόβλημα, επιστρέφει το αποτέλεσμα στον κατάλληλο πράκτορα που γνωρίζει πως θα το συνδυάσει με τα υπόλοιπα στη συνολική λύση.

---

[5] [Wooldridge 2002 κεφ.9]

Από τα παραπάνω μας ενδιαφέρει η ανάθεση υποπροβλημάτων στους πράκτορες. Για την πραγματοποίησή της υπάρχουν διάφοροι τρόποι διαπραγμάτευσης.

1.3. Διαπραγμάτευση μεταξύ πρακτόρων.

Διαπραγμάτευση[6] (negotiation) είναι μια μορφή αλληλεπίδρασης σύμφωνα με την οποία, μια ομάδα πρακτόρων με συγκρουόμενα συμφέροντα και επιθυμία για συνεργασία, προσπαθούν να έλθουν σε μια κοινά αποδεκτή συμφωνία για το μοίρασμα σπάνιων πόρων ή εργασιών. Ο όρος πόρος χρησιμοποιείται με την ευρεία έννοια, δηλαδή αναφέρεται σε υπηρεσίες, ευκολίες, χρόνο, χρήμα, κοκ.

Διαπραγμάτευση απαιτείται σε αρκετές περιπτώσεις: για την εγκατάσταση μιας κοινά αποδεκτής συμφωνίας μεταξύ εγωιστών πρακτόρων, κατά την παρακολούθηση της εκτέλεσης μιας συμφωνίας αν προκύψουν διαφωνίες από τις απέναντι πλευρές, για την αντιμετώπιση ασυνεπειών στην κατανεμημένη πληροφορία κοκ.

Η πολυπλοκότητα μιας διαπραγμάτευσης εξαρτάται από τον αριθμό των εμπλεκομένων πρακτόρων και μπορεί να είναι ένας-προς-ένα, πολλοί-προς-ένα ή πολλοί-προς-πολλούς. Εξαρτάται επίσης από τον αριθμό των ζητημάτων προς διαπραγμάτευση (single issue - multiple issue), π.χ. στην περίπτωση της αγοράς αυτοκινήτου ενδιαφέρουν η τιμή, η εγγύηση και τα παρεχόμενα αξεσουάρ. Μάλιστα σε κάποιες περιπτώσεις οι αντίθετες πλευρές δεν συμφωνούν καν για τη διαπραγμάτευση των ίδιων ζητημάτων. Ειδική περίπτωση διαπραγμάτευσης πολλοί-προς-ένα είναι οι δημοπρασίες, στις οποίες ενδιαφέρει ένα ζήτημα, η τιμή.

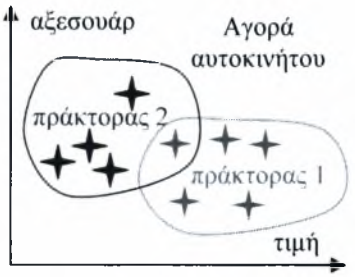
Μια διαπραγμάτευση συνήθως λαμβάνει χώρα σε γύρους (rounds). Σε κάθε γύρο ο ένας εμπλεκόμενος κάνει συγκεκριμένη πρόταση (proposal) την οποία ο άλλος μπορεί να απορρίψει (reject) ή να αποδεχτεί (accept). Σε περίπτωση απόρριψης, ο πρώτος θα κάνει νέα πρόταση, κοκ, μέχρι τελικά κάποια να γίνει αποδεκτή. Στην απλή περίπτωση η όλη διαδικασία μπορεί να αποδειχθεί ιδιαίτερα χρονοβόρα. Είναι πιο αποτελεσματικό να ασκήσει ο αντίπαλος κριτική (critique), δηλαδή σχόλιο για το ποιο τμήμα της πρότασης του αρέσει και ποιο όχι, ή να κάνει αντιπρόταση (counter-proposal) με βάση τις προτιμήσεις του.

Για να αναπαραστήσουμε γραφικά το χώρο των πιθανών προτάσεων, μπορούμε να αντιστοιχίσουμε κάθε ζήτημα της διαπραγμάτευσης σε μια διάσταση. Για παράδειγμα, στην αγορά αυτοκινήτου που ενδιαφέρουν η τιμή και

---

[6] [Wooldridge 2002 κεφ.7], [Jennings 2001]

τα παρεχόμενα αξεσουάρ, η απεικόνιση γίνεται στο διδιάστατο χώρο, όπως στο σχήμα 1.2.



Σχήμα 1.2

Κάθε διαπραγμάτευση έχει τα εξής συστατικά:

1. το *σύνολο διαπραγμάτευσης*, που αναπαριστά το χώρο των πιθανών προτάσεων. Περιγράφει τις τιμές για όλα τα ζητήματα της διαπραγμάτευσης που ενδιαφέρουν (π.χ. τιμή από 10€ έως 15€ και χρώμα μπλε ή πράσινο).
2. το *πρωτόκολλο* διαπραγμάτευσης, που είναι το σύνολο των κανόνων που διέπουν την επικοινωνία. Ορίζει τους αποδεκτούς συμμετέχοντες, τις πιθανές καταστάσεις (π.χ. η προσφορά έγινε αποδεκτή, η διαπραγμάτευση έκλεισε), τα γεγονότα που προκαλούν αλλαγή κατάστασης (π.χ. δεν υπάρχουν άλλες προσφορές), τις νόμιμες ενέργειες των συμμετεχόντων σε συγκεκριμένες καταστάσεις (π.χ. ποια μηνύματα μπορούν να σταλούν, από ποιον, σε ποιον και πότε).
3. τις *στρατηγικές* των πρακτόρων, που είναι μοντέλα λήψης απόφασης για το πώς αποτιμούν τις προτάσεις που λαμβάνουν και το πώς επιλέγουν τις προτάσεις που υποβάλλουν, έτσι ώστε να πετύχουν το στόχο τους. Οι στρατηγικές μπορεί να είναι διαφορετικές για κάθε πράκτορα και μυστικές.
4. ένα *κανόνα*, που ορίζει πότε σταματά η διαπραγμάτευση και ποια είναι η τελική συμφωνία.

Τα πρωτόκολλα είναι κατά κανόνα έτσι σχεδιασμένα, ώστε να παρέχουν συγκεκριμένες επιθυμητές ιδιότητες, όπως θα δούμε παρακάτω. Υπάρχουν διαφορετικά πρωτόκολλα και στρατηγικές διαπραγμάτευσης, που το καθένα μπορεί να είναι αποδοτικό κάτω από συγκεκριμένες συνθήκες. Παρόλα αυτά, για να γίνει δυνατή η αυτοματοποίηση των διαπραγματεύσεων, έχει προταθεί και μελετηθεί μια σειρά τεχνικών, βασισμένες στη θεωρία παιγνίων, σε ευρετικές τεχνικές και σε επιχειρήματα.

Η *διαπραγμάτευση βασισμένη στη θεωρία παιγνίων*[7] (game-theoretic) είναι θεωρητική μέθοδος που προέρχεται από τον κλάδο των Οικονομικών. Μελετά

[7] [Jennings 2001], [Rahwan 2003b], [Faratin 2000]

κυρίως την αλληλεπίδραση μεταξύ εγωιστών πρακτόρων και αφορά τόσο το πρωτόκολλο διαπραγμάτευσης, όσο και τη στρατηγική απόφασης πράκτορα. Αναλύει την αλληλεπίδραση σαν παιχνίδι στρατηγικής μεταξύ όμοιων συμμετεχόντων και αναζητά το σημείο ισορροπίας. Εντοπίζει τη βέλτιστη δυνατή λύση μέσα από το σύνολο των πιθανών λύσεων, αλλά όχι απαραίτητα το πώς θα τη βρούμε.

Οι κλασικές προσεγγίσεις αυτού του τύπου έχουν σημαντικούς περιορισμούς. Καταρχήν, δεν έχει βρεθεί ένα γενικό μοντέλο που να καλύπτει όλες τις περιπτώσεις. Επίσης, η μέθοδος είναι χρονοβόρα και απαιτεί πολλούς υπολογιστικούς πόρους στους πράκτορες. Απαιτείται ο χώρος των αποτελεσμάτων να είναι πεπερασμένος και πλήρως γνωστός, κάτι που στην πράξη δεν ισχύει πάντα. Απαιτείται οι πράκτορες να σκέφτονται και να ενεργούν ορθολογικά (rationally), που επίσης δεν είναι πάντα δυνατό. Μάλιστα στην πράξη οι πράκτορες μπορεί ακόμη και να περιέχουν σφάλματα κώδικα ή να είναι κακόβουλοι, γεγονός που προκαλεί επιπλέον προβλήματα.

Παρακάτω θα μελετηθούν τεχνικές βασισμένες στη θεωρία παιγνίων, όπως είναι το μονοτονικό πρωτόκολλο συμβιβασμού και η στρατηγική Zeuthen.

Στη *διαπραγμάτευση βασισμένη σε ευρετικές τεχνικές*[8] (heuristic-based) πραγματοποιείται μη εξαντλητική αναζήτηση στο χώρο των λύσεων, οπότε το αποτέλεσμα προκύπτει με μικρότερο κόστος. Με τον τρόπο αυτό παράγονται ικανοποιητικές αλλά όχι βέλτιστες λύσεις. Οι τεχνικές βασίζονται κυρίως σε εμπειρικούς ελέγχους και εκτιμήσεις και κάποιες φορές σε προσεγγίσεις των μεθόδων θεωρίας παιγνίων. Τα μοντέλα αυτά απαιτούν λιγότερους πόρους και ορθολογικότητα από την πλευρά των πρακτόρων. Βασίζονται σε ρεαλιστικές υποθέσεις, οπότε μπορούν ευκολότερα να αυτοματοποιηθούν και να χρησιμοποιηθούν στην πράξη σε μεγαλύτερο πλήθος εφαρμογών. Σημειώνεται όμως ότι δεν μπορεί το ίδιο μοντέλο να χρησιμοποιηθεί σε όλους τους τομείς εφαρμογών (είναι domain-dependent).

Ενώ οι ευρετικές μέθοδοι προσπαθούν να ξεπεράσουν κάποια από τα προβλήματα της θεωρίας παιγνίων, ωστόσο παρουσιάζουν μια σειρά μειονεκτημάτων. Ειδικότερα, το προκύπτον αποτέλεσμα είναι συχνά υπο-βέλτιστο, εφόσον δεν γίνονται υποθέσεις ορθολογικότητας και δεν εξετάζεται ο πλήρης χώρος των πιθανών αποτελεσμάτων. Επίσης, είναι δύσκολο να προβλεφθεί η συμπεριφορά του συστήματος, οπότε χρειάζεται ακόμη εκτενής ανάλυση των μοντέλων μέσω προσομοιώσεων και εμπειρικών αναλύσεων.

[8] [Jennings 2001], [Rahwan 2003b], [Faratin 2000]

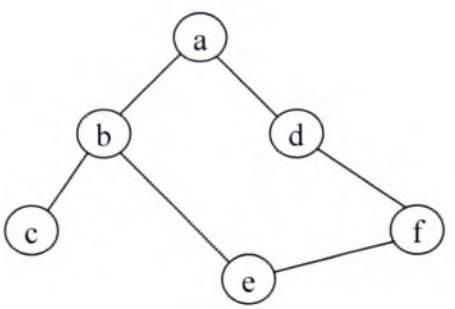
Τέλος, έχουν προταθεί διάφορες ευρετικές μέθοδοι και δεν είμαστε σίγουροι ποια από τις υπάρχουσες να εφαρμόσουμε σε κάθε περίπτωση.

Συμπερασματικά, δεν υπάρχει μια καθολική προσέγγιση για τη διαπραγμάτευση μεταξύ πρακτόρων που να ταιριάζει σε κάθε εφαρμογή και οποιοδήποτε πρόβλημα. Υπάρχει ένα σύνολο προσεγγίσεων, που η κάθε μια βασίζεται σε διαφορετικές υποθέσεις για το περιβάλλον και τους εμπλεκόμενους. Στη συνέχεια, επειδή η διαπραγμάτευση έχει άμεση σχέση με τη διαμοίραση εργασιών, θα εστιάσουμε στην εφαρμογή της σε τομείς προσανατολισμένους σε εργασίες.

1.4. Τομείς προσανατολισμένοι σε εργασίες.

Οι *τομείς προσανατολισμένοι σε εργασίες*[9] - ΤΠΕ (task oriented domains) είναι περιοχές προβλημάτων στις οποίες η δραστηριότητα των πρακτόρων μπορεί να οριστεί ως μια ακολουθία εργασιών τις οποίες πρέπει να πραγματοποιήσουν. Μια *εργασία* (task) είναι μια αδιαίρετη δουλειά που πρέπει να επιτευχθεί από κάποιον πράκτορα. Παρόλο που οι πράκτορες έχουν όλους τους πόρους για να υλοποιήσουν τις ατομικές τους εργασίες, μπορεί να ωφεληθούν από την ανακατανομή κάποιων από αυτές. Κλασικά παραδείγματα ΤΠΕ είναι η αποστολή φαξ, η παράδοση αλληλογραφίας, η αναζήτηση σε βάση δεδομένων, κ.α.

Στον τομέα αποστολής φαξ (fax domain) οι πράκτορες έχουν να στείλουν μηνύματα με φαξ σε τοποθεσίες ενός τηλεφωνικού δικτύου που αναπαρίσταται με ένα γράφο με βάρη, όπως στο σχήμα 1.3. Ο πράκτορας βρίσκεται στον κόμβο αποστολής και για να στείλει φαξ πρέπει να συνδεθεί με τον κόμβο παραλαβής. Το κόστος σύνδεσης εξαρτάται από την απόσταση των δύο κόμβων. Αφού



Σχήμα 1.3

[9] [Rosenschein 1994, κεφ.2,3]

συνδεθεί, μπορεί να στείλει όσα φαξ θέλει, χωρίς επιπλέον κόστος. Επίσης, οι πράκτορες μπορούν χωρίς κόστος να ανταλλάσουν μεταξύ τους μηνύματα προς αποστολή. Για παράδειγμα, στο σχήμα, αν δυο πράκτορες έχουν να στείλουν φαξ από τον κόμβο a στους b και f, τότε συμφέρει να συνεννοηθούν και να στείλει ο ένας μόνο στον κόμβο b και ο άλλος μόνο στον f.

Στον τομέα αλληλογραφίας (postmen domain) οι πράκτορες έχουν να μοιράσουν γράμματα σε τοποθεσίες που αναπαρίστανται με ένα γράφο όπως του σχήματος 1.3 και μετά να επιστρέψουν στο ταχυδρομείο. Κόστος μιας εργασίας είναι το μήκος του μικρότερου μονοπατιού μέχρι τον προορισμό και κατόπιν πίσω στον αρχικό κόμβο. Για παράδειγμα, αν ο ένας πράκτορας έχει να παραδώσει στους κόμβους d, f, b και ο άλλος στους b, e, τότε συμφέρει να επισκεφτεί ο ένας τους d, f και ο άλλος τους b, e. Παρόμοιο πρόβλημα αποτελεί ο τομέας παράδοσης (delivery domain) στον οποίο οι πράκτορες δεν χρειάζεται να επιστρέψουν στον αρχικό κόμβο.

Στον τομέα αναζήτησης σε βάση δεδομένων (database domain) οι πράκτορες εκτελούν σύνθετα ερωτήματα, τα οποία πιθανώς να παρουσιάζουν επικαλύψεις. Κόστος μιας εργασίας είναι το ελάχιστο πλήθος των απαιτούμενων πράξεων του συστήματος διαχείρισης της βάσης. Για παράδειγμα, έστω ότι ο ένας πράκτορας αναζητά βιβλιογραφία με αντικείμενο «πολυπρακτορικά συστήματα» νεότερη του 2000 και ο άλλος «πολυπρακτορικά συστήματα» με συγγραφέα το Wooldridge. Τα δυο πρώτα υποερωτήματα είναι κοινά και θα μπορούσε να τα εκτελέσει μόνο ο ένας.

Τυπική διατύπωση ΤΠΕ.

Ένας *ΤΠΕ*[10] ορίζεται ως μια τριάδα <T,A,c>, όπου:

(1) $T = \{t_0,t_1,\ldots t_{N-1}\}$ είναι το σύνολο όλων των διαθέσιμων εργασιών.

(2) $A = \{A_0,\ldots,A_{K-1}\}$ είναι μια διατεταγμένη λίστα πρακτόρων. Στη συνέχεια, θα θεωρήσουμε ότι έχουμε δύο πράκτορες. Ο συμβολισμός για τον πράκτορα είναι A_k με $k = 0$ ή 1.

(3) c είναι μια μονοτονική συνάρτηση $[2^T]\rightarrow R^+$, όπου $[2^T]$ αναπαριστά όλα τα πεπερασμένα υποσύνολα του T. Για κάθε πεπερασμένο υποσύνολο εργασιών X, c(X) είναι το κόστος της εκτέλεσης των εργασιών του X από ένα πράκτορα. Η c είναι μονοτονική, δηλαδή αν $X\subseteq Y\subseteq T$, τότε $c(X)<= c(Y)$. Ισχύει $c(\varnothing)=0$.

Μια *αλληλεπίδραση* (encounter) σε ένα ΤΠΕ είναι μια διατεταγμένη λίστα $e = (T_0,T_1)$, τέτοια ώστε κάθε T_k είναι ένα πεπερασμένο υποσύνολο του T που ο

[10] [Rosenschein 1994, κεφ.3], [Wooldridge 2002, κεφ.7]

πράκτορας A_k πρέπει να πραγματοποιήσει. Το T_k λέγεται και *στόχος* (goal) του A_k. Το κόστος μιας αλληλεπίδρασης e για ένα πράκτορα A_k είναι το κόστος εκτέλεσης των εργασιών του T_k, δηλαδή $c_k(e) = c(T_k) = \sum c(t_i)$, $\forall t_i \in T_k$.

Μια *πρόταση* (pure deal) είναι η ανακατανομή των εργασιών μεταξύ των πρακτόρων, δηλαδή μια διατεταγμένη λίστα $d = (D_0,D_1)$, τέτοια ώστε $D_0 \cup D_1 = T_0 \cup T_1$. Κάθε πράκτορας A_k δεσμεύεται να εκτελέσει τις εργασίες D_k με κόστος $c_k(d) = c(D_k) = \sum c(t_i)$, $\forall t_i \in D_k$.

Πρόταση σύγκρουσης (conflict deal) λέγεται η τελική αλληλεπίδραση εργασιών που προκύπτει αν η διαπραγμάτευση καταλήξει σε αποτυχία. Η πρόταση σύγκρουσης Θ σε αυτή την περίπτωση συμπίπτει με την αρχική αλληλεπίδραση εργασιών, δηλαδή $\Theta \equiv (T_0,T_1)$.

Ορίζουμε *χρησιμότητα* (utility) ως $u_k(\delta) \equiv c(T_k) - c(D_k)$. Η χρησιμότητα δείχνει πόσο «καλή» είναι η πρόταση d για τον πράκτορα A_k. Αρνητική χρησιμότητα σημαίνει ότι ο πράκτορας βρίσκεται σε χειρότερη θέση από ότι στην αρχική αλληλεπίδραση εργασιών.

Λέμε ότι ο πράκτορας A_k *προτιμά* τη d τουλάχιστον όσο τη d' και συμβολίζουμε $d \succeq_k d'$, αν ισχύει $u_k(d) \geq u_k(d')$. Παρόμοια, λέμε ότι ο πράκτορας A_k *αυστηρά προτιμά* τη d έναντι της d' και συμβολίζουμε $d \succ_k d'$, αν ισχύει $u_k(d) > u_k(d')$.

Λέμε ότι η d *κυριαρχεί* (dominates) της d' και συμβολίζουμε $d \succ d'$, αν και μόνο αν για κάθε πράκτορα A_k ισχύει $u_k(d) \geq u_k(d')$ και επιπλέον υπάρχει κάποιος πράκτορας i για τον οποίο ισχύει $u_i(d) > u_i(d')$. Στην περίπτωση αυτή τη d την προτιμούν όλοι οι πράκτορες έναντι της d'.

Λέμε ότι η d *κυριαρχεί ασθενώς* (weakly dominates) της d' ($d \succeq d'$), αν και μόνο αν $\forall A_k$ ισχύει $u_k(d) \geq u_k(d')$.

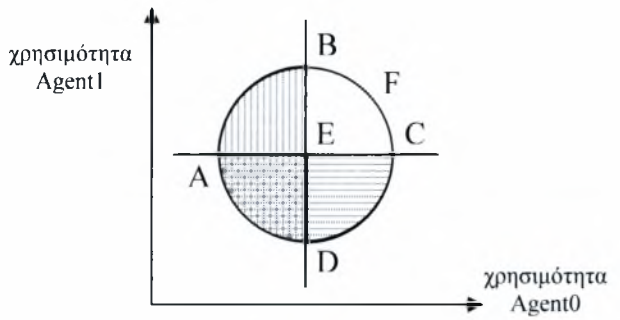
Λέμε ότι η d είναι *ισοδύναμη* (equivalent) της d' ($d \equiv d'$), αν και μόνο αν $\forall A_k$ ισχύει $u_k(d) = u_k(d')$.

Μια πρόταση d λέγεται *pareto βέλτιστη* (pareto optimal), αν δεν υπάρχει άλλη d' τέτοιο ώστε η d να κυριαρχεί της d΄ ($d \succ d'$). Αν μια πρόταση είναι pareto βέλτιστη, δεν υπάρχει άλλη εναλλακτική που να βελτιώνει τη χρησιμότητα ενός πράκτορα, παρά μόνο με κόστος για τον αντίπαλο.

Μια πρόταση d λέγεται *ατομικά ορθολογική* (individual rational), αν κυριαρχεί (ενδεχομένως ασθενώς) της πρότασης σύγκρουσης ($d \succeq \Theta$). Η d είναι ατομικά ορθολογική αν δίνει σε όλους τους πράκτορες μη αρνητική χρησιμότητα.

Σύνολο διαπραγμάτευσης (negotiation set) είναι το σύνολο όλων των προράσεων που είναι ατομικά ορθολογικές και pareto βέλτιστες.

Σημειώνουμε ότι προτάσεις που αναθέτουν την ίδια εργασία και στους δυο πράκτορες ταυτόχρονα δεν ανήκουν στο σύνολο διαπραγμάτευσης, γιατί δεν είναι pareto βέλτιστες. Κι αυτό γιατί υπάρχει καλύτερη πρόταση για τον ίδιο τον πράκτορα που να μην είναι χειρότερη για τον άλλο. Για παράδειγμα, η <{t0,t1}{t0,t2}> δεν είναι pareto βέλτιστη, ενώ η <{t1}{t0,t2}> είναι.



Σχήμα 1.4

Ο χώρος όλων των πιθανών προτάσεων μπορεί να αναπαρασταθεί όπως φαίνεται στο σχήμα 1.4. Το σημείο Ε αντιστοιχεί στην αρχική αλληλεπίδραση και την πρόταση σύγκρουσης.

Σύμφωνα με το κριτήριο της ατομικής ορθολογικότητας, ο πράκτορας Agent0 προτιμά τις προτάσεις που βρίσκονται στο ημικύκλιο BCD με χρησιμότητα τουλάχιστον όσο το Ε. Ομοίως, ο πράκτορας Agent1 προτιμά το ημικύκλιο ABC. Η τομή τους, δηλαδή οι προτάσεις που συμφέρουν και τους δυο, είναι το τεταρτημόριο EBC.

Pareto βέλτιστες είναι οι προτάσεις που βρίσκονται πάνω στο τόξο BC, γιατί μόνο για αυτές δεν υπάρχουν άλλες προτάσεις καλύτερες για τον ένα πράκτορα, που να μην είναι χειρότερες για τον άλλο. Οπότε το ζητούμενο στη διάρκεια της διαπραγμάτευσης είναι να βρεθούν αυτές οι προτάσεις, καθώς και εκείνες που αντιστοιχούν στα αρχικά σημεία B, C.

1.5. Μονοτονικό πρωτόκολλο συμβιβασμού.

Υπάρχουν διάφορα πρωτόκολλα που δίνουν τους κανόνες για τη διαδικασία της διαπραγμάτευσης. Ένα από αυτά είναι το *μονοτονικό πρωτόκολλο συμβιβασμού*[11] – ΜΠΣ (monotonic concession protocol), σύμφωνα με το οποίο:

- η διαπραγμάτευση προχωρά σε γύρους

[11] [Rosenschein 1994, κεφ.3]

- στον πρώτο γύρο και οι δύο πράκτορες κάνουν ταυτόχρονα μια πρόταση από το σύνολο διαπραγμάτευσης. Ο πράκτορας A_M προτείνει d_M και ο A_O προτείνει d_O.
- συμφωνία (agreement) επέρχεται αν ένας από τους δύο πράκτορες βρει την πρόταση του αντιπάλου τουλάχιστον τόσο καλή για τον ίδιο όσο και τη δική του. Συγκεκριμένα, αν $u_M(d_O) \geq u_O(d_M)$ ή $c_O(d_M) \geq c_M(d_O)$, ο A_M αποδέχεται τη d_O. Αν και οι δυο αποδεχτούν την πρόταση του αντιπάλου, τότε επιλέγεται μια με βάση κάποιο κριτήριο.
- αν δεν επέλθει συμφωνία, τότε γίνεται νέος γύρος ταυτόχρονων προτάσεων. Κάθε πράκτορας A_M μπορεί να κάνει πρόταση με την ίδια ή μεγαλύτερη χρησιμότητα για τον αντίπαλο από αυτή που πρότεινε στον προηγούμενο γύρο, δηλαδή να παραμείνει στη θέση του (stand still) ή να υποχωρήσει (concede).
- αν σε κάποιο γύρο δεν υποχωρήσει κανείς πράκτορας, τότε η διαπραγμάτευση τερματίζει με σύγκρουση (conflict).

Ας ξαναδούμε το χώρο διαπραγμάτευσης του σχήματος 1.4. Αρχικό σημείο διαπραγμάτευσης για τον Agent1 είναι το σημείο B, που αντιστοιχεί για τον ίδιο σε πρόταση με μέγιστη χρησιμότητα και για τον αντίπαλο σε χρησιμότητα όση και της αρχικής αλληλεπίδρασης. Τελικό σημείο είναι το C, με χρησιμότητα για τον ίδιο όση και της αρχικής αλληλεπίδρασης και για τον αντίπαλο μέγιστη χρησιμότητα. Για τον Agent0, από την άλλη, αρχικό σημείο είναι το C και τελικό το B.

Κατά τη διαπραγμάτευση με το ΜΠΣ, καθένας ξεκινά από το αρχικό του σημείο και υποχωρεί σταδιακά μέχρι, είτε να προκύψει συμφωνία στο σημείο F, όταν η πρόταση του αντιπάλου έχει μεγαλύτερη χρησιμότητα για τον ίδιο από τη δική του, είτε να εγκαταλειφθεί η διαπραγμάτευση, επειδή δεν υποχώρησε κανείς ή επειδή φτάσαμε το τελικό σημείο. Στην περίπτωση αυτή επιστρέφουμε στο σημείο E της αρχικής αλληλεπίδρασης.

Μελετώντας το ΜΠΣ, προκύπτουν κάποια ζητήματα[12] όσον αφορά τις αποφάσεις που παίρνει κάθε πράκτορας.

I. Ποια είναι η πρώτη πρόταση του κάθε πράκτορα;

Πρώτη πρόταση θα έπρεπε να είναι αυτή που προτιμά ο πράκτορας περισσότερο από το σύνολο διαπραγμάτευσης.

[12] [Rosenschein 1994, κεφ.3], [Wooldridge 2002, κεφ.7]

II. Πως θα βρει κανείς την αμέσως επόμενη καλύτερη πρόταση;

Το ερώτημα αυτό είναι σημαντικό και αρκετά πολύπλοκο, δεδομένου ότι υπάρχουν συνολικά $2^{|T|}$ πιθανές προτάσεις. Σε πραγματικά παραδείγματα ίσως να μην είναι πάντοτε πρακτικά δυνατός ο υπολογισμός του κόστους $2^{|T|}$ φορές.

III. Ποιος πρέπει να υποχωρήσει σε κάθε γύρο;

Η *στρατηγική Zeuthen*[13] είναι ένας τρόπος που προτείνεται από τη βιβλιογραφία και υποδεικνύει ποιος πράκτορας να υποχωρήσει κάθε φορά. Σύμφωνα με αυτή, σε κάθε γύρο t και οι δυο πράκτορες A_M και A_O υπολογίζουν το ρίσκο r_M και r_O, δηλαδή το βαθμό προθυμίας να ρισκάρουν σύγκρουση.

$$r_M^t = \frac{\text{χρησιμότητα για τον } A_M \text{ αν υποχωρήσει και αποδεχτεί την προσφορά του } A_O}{\text{χρησιμότητα για τον } A_M \text{ αν δεν υποχωρήσει και προκαλέσει σύγκρουση}}$$

$$r_M^t = \begin{cases} 1 & \text{αν } u_M(\delta_M^t) = 0 \\ \dfrac{u_M(\delta_M^t) - u_k(\delta_O^t)}{u_M(\delta_O^t)} & \text{αλλιώς} \end{cases}$$

Αν t δεν είναι ο τελικός γύρος διαπραγμάτευσης το ρίσκο είναι μεταξύ 0 και 1, αλλιώς τουλάχιστο ένας έχει αρνητικό ρίσκο. Η στρατηγική προτείνει συμβιβασμό όταν το υπολογισμένο ρίσκο βρεθεί μικρότερο ή ίσο από αυτό του αντιπάλου. Αυτό σημαίνει ότι όταν οι δύο πράκτορες έχουν σ' ένα βήμα ίδιο ρίσκο υποχωρούν και οι δυο, διαφορετικά υποχωρεί μόνο ο ένας.

Ας θεωρήσουμε την περίπτωση όπου κατά τον τελευταίο γύρο διαπραγμάτευσης και οι δυο πράκτορες βρίσκονται με ίσο ρίσκο, οπότε κανονικά θα έπρεπε και οι δυο να υποχωρήσουν. Τότε μπορεί ο ένας να αποπειραθεί να 'κλέψει' και να μην υποχωρήσει. Αν και οι δυο κάνουν το ίδιο, τότε θα καταλήξουν σε σύγκρουση. Για να αποφευχθεί το προηγούμενο, εισάγεται η *επεκτεταμένη στρατηγική Zeuthen* (extended Zeuthen strategy). Σύμφωνα με αυτή, αν στο τελευταία βήμα βρεθεί ίσο ρίσκο και στους δυο, τότε υπάρχουν δυο παραλλαγές: α) επιλέγεται τυχαία από τον ένα ή από το σύστημα το ποιος από τους δύο θα υποχωρήσει[14] και β) επιλέγει τυχαία ο καθένας αν θα υποχωρήσει ή όχι[15].

IV. Πόσο πρέπει να υποχωρήσει κανείς;

Ικανή υποχώρηση είναι αυτή που αλλάζει την ισορροπία του ρίσκου μεταξύ των δυο πρακτόρων, έτσι ώστε στον επόμενο γύρο να αναγκαστεί ο αντίπαλος να υποχωρήσει. *Ελάχιστη ικανή υποχώρηση* (minimal sufficient concession) είναι η ικανή υποχώρηση που δίνει στον αντίπαλο τη μικρότερη χρησιμότητα.

[13] [Rosenschein 1994, κεφ.3], [Wooldridge 2002, κεφ.7]
[14] [Wooldridge 2002, κεφ.7]
[15] [Rosenschein 1994, κεφ.3]

Στη συνέχεια, θα πραγματοποιηθεί αποτίμηση[16] του ΜΠΣ. Επιθυμητές ιδιότητες για ένα πρωτόκολλο γενικά είναι οι παρακάτω:

- *αποτελεσματικότητα* (efficiency): δηλαδή με την επίτευξη συμφωνίας στο σύστημα να μην σπαταλάται χρησιμότητα. Εδώ αναφέρονται η pareto βελτιστοποίηση, που είδαμε νωρίτερα, και η καθολική βελτιστοποίηση, που επιτυγχάνεται όταν μεγιστοποιείται το άθροισμα των χρησιμοτήτων των συμμετεχόντων
- *σταθερότητα* (stability): όταν κανείς πράκτορας δεν βρίσκει κίνητρο να παρεκκλίνει από τις συμφωνημένες στρατηγικές. Σχετικός είναι και ο όρος Nash equilibrium. Δυο στρατηγικές s και s΄ βρίσκονται σε ισορροπία Nash όταν, αν υποθέσουμε ότι ένας πράκτορας χρησιμοποιεί την s, τότε ο αντίπαλος δεν μπορεί να βρει κάτι καλύτερο παρά να χρησιμοποιήσει την s΄ και αντίστροφα
- *απλότητα* (simplicity): ένα απλό πρωτόκολλο κάνει προφανή στον πράκτορα την κατάλληλη στρατηγική. Ένα απλό σύστημα έχει σχετικά μικρές υπολογιστικές και επικοινωνιακές απαιτήσεις. Η απλότητα σχετίζεται με την αποτελεσματικότητα και τη σταθερότητα
- *κατανομή* (distribution): όταν οι αποφάσεις και οι υπολογισμοί γίνονται τοπικά και μειώνεται η επικοινωνία και η ανάγκη ύπαρξης ενός κεντρικού κόμβου
- *συμμετρία* (symmetry): δηλαδή να αντιμετωπίζονται οι πράκτορες με παρόμοιο τρόπο, κανείς να μην παίζει ιδιαίτερο ρόλο
- *εγγυημένη επιτυχία* (guaranteed success): αν εγγυάται ότι θα επιτευχθεί συμφωνία
- *ατομική ορθολογικότητα* (individual rationality): αν η τήρηση των κανόνων είναι στα άμεσα συμφέροντα των συμμετεχόντων

Το ΜΠΣ μοιάζει αρκετά στον τρόπο που διαπραγματεύονται μεταξύ τους οι άνθρωποι. Κατά πόσο όμως σε συνδυασμό με τη στρατηγική Zeuthen ικανοποιεί τα παραπάνω κριτήρια;

Καταρχήν, κάνουμε την παρατήρηση ότι δεν ορίζεται από το πρωτόκολλο να χρησιμοποιούν και οι δυο πράκτορες τις ίδιες βασικές λειτουργίες, ούτε είναι απαραίτητο να γνωρίζει ο καθένας αυτές που χρησιμοποιεί ο αντίπαλος. Στη βιβλιογραφία, όμως, αναφέρεται ότι με τη χρήση στρατηγικών που χαρακτηρίζονται από σταθερότητα, δεν έχει νόημα η μυστικότητα από την πλευρά των προγραμματιστών, αλλά είναι επιθυμητό να γνωρίζουν και να

[16] [Jennings 2001], [Rosenschein 1994, κεφ.3], [Wooldridge 2002, κεφ.7]

υιοθετούν τη στρατηγική του αντιπάλου για αποφυγή ασυμβατοτήτων και μεγαλύτερη απόδοση.

Με το ΜΠΣ δεν εξασφαλίζεται επιτυχία, αλλά εξασφαλίζεται τερματισμός μετά από πεπερασμένο αριθμό γύρων. Το πρωτόκολλο είναι συμμετρικό, ενώ είναι κατανεμημένο και δεν απαιτείται κεντρικός συντονιστής. Δεν εγγυάται μεγιστοποίηση κοινωνικής ωφέλειας. Αντίθετα, εγγυάται ότι, αν είναι διαθέσιμες όλες οι προτάσεις του συνόλου διαπραγμάτευσης και δεν έχουν χρησιμοποιηθεί ευρετικές και προσεγγιστικές τεχνικές, τότε, αν επιτευχθεί συμφωνία, το αποτέλεσμα θα είναι ατομικά ορθολογικό και pareto βέλτιστο. Έχει αποδειχθεί[17] ότι αν και οι δυο πράκτορες χρησιμοποιούν Zeuthen, τότε θα συμφωνήσουν σε μια πρόταση που μεγιστοποιεί το γινόμενο των χρησιμοτήτων τους.

Αναφορικά με τη σταθερότητα, η απλή Zeuthen στρατηγική δεν είναι σταθερή, όπως φαίνεται στην περίπτωση που είδαμε, που και οι δυο πράκτορες αποδέχονται στον τελευταίο γύρο την πρόταση του αντιπάλου. Αντίθετα, η επεκτεταμένη στρατηγική Zeuthen είναι σταθερή[18].

Από την άλλη, όταν στην επεκτεταμένη στρατηγική Zeuthen προκύψει το πρόβλημα του τελευταίου βήματος και επιλέγει καθένας τυχαία αν θα υποχωρήσει ή όχι, υπάρχει πιθανότητα να μην υποχωρήσει κανείς και να καταλήξουν σε σύγκρουση. Τότε θα επικρατήσει η πρόταση σύγκρουσης που δεν είναι απαραίτητα pareto βέλτιστη. Η στρατηγική αυτή συνεπώς δεν είναι πάντα αποτελεσματική. Σημειώνεται ότι το πρόβλημα αυτό δεν παρουσιάζεται όταν επιλέγεται τυχαία από τον ένα ή από το σύστημα το ποιος από τους δύο θα υποχωρήσει.

Όσον αφορά την απλότητα, ενδέχεται να απαιτηθούν $O(2^{|T|})$ υπολογισμοί της συνάρτησης κόστους, καθώς και $O(2^{|T|})$ γύροι που συνεπάγονται μεγάλο επικοινωνιακό κόστος. Αυτό αποτελεί σημαντικό μειονέκτημα για το ΜΠΣ. Στη συνέχεια, θα εξετάσουμε υπολογιστικά πώς μπορεί να βελτιωθεί η υλοποίηση του πρωτοκόλλου.

Ανακεφαλαίωση.

Στο κεφάλαιο αυτό, παρέχεται το βασικό υπόβαθρο γνώσεων απαραίτητο για τη συνέχεια. Έγινε περιγραφή της συνεργατικής κατανεμημένης επίλυσης προβλήματος (ΣΚΕΠ). Ορίστηκαν οι βασικές έννοιες των πρακτόρων και πολυπρακτορικών συστημάτων. Πραγματοποιήθηκε περιγραφή και αναφορά στα

[17] [Rosenschein 1994, κεφ.3]
[18] [Rosenschein 1994, κεφ.3]

είδη διαπραγμάτευσης μεταξύ πρακτόρων. Έγινε τυπική διατύπωση των τομέων προσανατολισμένων σε εργασίες (ΤΠΕ). Παρουσιάστηκε το μονοτονικό πρωτόκολλο συμβιβασμού (ΜΠΣ) και έγινε αναφορά στα ερωτήματα που προκύπτουν σχετικά με τις βασικές λειτουργίες των πρακτόρων κατά την εφαρμογή του πρωτοκόλλου.

Στο κεφάλαιο που ακολουθεί, θα πραγματοποιηθεί λεπτομερέστερη ανάλυση της λειτουργίας του πρωτοκόλλου και των στρατηγικών των πρακτόρων κατά τη διάρκεια της διαπραγμάτευσης. Θα συζητηθούν εναλλακτικές λύσεις και θα γίνει απόπειρα υπολογιστικών βελτιώσεων.

2.1. Είδη ατομικών εργασιών.

Στο σημείο αυτό, πριν μπούμε σε λεπτομέρειες του ΜΠΣ και των βασικών λειτουργιών των πρακτόρων, είναι καλό να μελετήσουμε καλύτερα το είδος της συνάρτησης κόστους σ' ένα ΤΠΕ και το πως μέσω αυτής κατατάσσουμε τις εργασίες σε είδη.

Η πιο απλή περίπτωση είναι να θεωρήσουμε ότι το κόστος είναι ένας σταθερός αριθμός, δηλαδή οι εργασίες $\{t_0,t_1,\ldots t_{N-1}\}$ έχουν κόστος τις σταθερές $\{c_0,c_1,\ldots c_{N-1}\}$. Για παράδειγμα, στο fax domain όπως περιγράφηκε στο προηγούμενο κεφάλαιο, για την αποστολή ενός φαξ απαιτείται μόνο το κόστος σύνδεσης. Έτσι, η αποστολή ενός ή περισσότερων φαξ με μια σύνδεση στη Λάρισα έχει κόστος 10, στην Αθήνα κόστος 20, στο Λονδίνο 50, στο Πεκίνο 100, κοκ. Τις εργασίες αυτού του τύπου τις ονομάζουμε *απλές* (simple). Στην περίπτωση αυτή μπορούν να γίνουν αρκετές απλοποιήσεις στις υποθέσεις που γίνονται και στη διαδικασία που θα ακολουθηθεί, όπως θα δούμε παρακάτω.

Από την άλλη, θα μπορούσαμε να θεωρήσουμε ότι το κόστος αποστολής φαξ εξαρτάται όχι μόνο από τη σύνδεση αλλά και από των αριθμό των σελίδων που αποστέλλονται, για παράδειγμα η αποστολή φαξ στο Λονδίνο να κοστίζει 50 για την πρώτη σελίδα και 3 για κάθε νέα σελίδα. Οι εργασίες αυτές στις οποίες το κόστος εξαρτάται από το περιεχόμενό τους, δηλαδή από το είδος και την ποσότητα, τις ονομάζουμε *σύνθετες* (complex). Συνήθως, το κόστος της εργασίας f_i(είδος, ποσότητα) είναι μια συνάρτηση της μορφής c_i(είδος, ποσότητα) = c_{i0}(είδος) + ποσότητα * c_{i1}(είδος). Στο σημείο αυτό κάνουμε την παρατήρηση ότι για να μην έχουμε να διαχειριστούμε δυο μεταβλητές μπορούμε να κάνουμε την εξής σύμβαση: να αναλύσουμε την f(είδος, ποσότητα) στις f_είδος$_j$(ποσότητα) και να έχουμε μόνο μία μεταβλητή. Π.χ. {φαξ_Λάρισα(10), φαξ_Αθήνα(10), φαξ_Λονδίνο(10), φαξ_Πεκίνο(2)}. Πλέον η συνάρτηση κόστους γίνεται $c_i(n) = c_{i0}+n*c_{i1}$, όπου c_{i0} και c_{i1} σταθερές.

Μια σύνθετη εργασία f(n) μπορεί να αναλυθεί σε n υποεργασίες που η κάθε μια έχει κόστος που δεν εξαρτάται από το περιεχόμενό της, δηλαδή είναι απλή. Οι υποεργασίες είναι οι f(1), f΄(1), f΄(1), …, f΄(1), ή πιο απλά f, f΄, f΄, …, f΄, με κόστος c_0+c_1, c_1, …, c_1 αντίστοιχα. Χρειάζεται όμως προσοχή στο γεγονός ότι η υποεργασία f΄(1) δεν μπορεί να σταθεί μόνη της, αλλά μόνο μαζί με την f(1), την πρώτη υποεργασία της ίδιας εργασίας. Για παράδειγμα, δυνατές περιπτώσεις είναι: {f(1)}, {f(1),f΄(1)}, {f(1),f΄(1),f΄(1)}, {f(1),f΄(1),f΄(1),f΄(1)}, κοκ. Η f θα

ονομάζεται *κύρια* (master) και οι f΄ *δευτερεύουσες* (slave). Οι εργασίες οι οποίες παρουσιάζουν εξαρτήσεις μεταξύ τους ονομάζονται *εξαρτημένες* (interdependent) .

Στην πράξη είναι δυνατόν στον ίδιο τομέα να υπάρχει πληθώρα εργασιών απλών και σύνθετων. Για παράδειγμα, αναφορικά με το fax domain μπορεί να θεωρήσουμε ότι έχει γίνει ειδική σύνδεση με Υπουργείο ώστε για την αποστολή φαξ να απαιτείται μόνο το κόστος σύνδεσης. Έτσι, έχουμε τις εργασίες {φαξ_Αθήνα(5), φαξ_Λάρισα(7), φαξ_Υπουργείο} με κόστος {$20+2n_1$, $10+n_2$, 5}. Στην περίπτωση λοιπόν που στον τομέα μας υπάρχουν σύνθετες εργασίες, μπορεί να εφαρμοστεί η εξής διαδικασία: α) αποσύνθεση των σύνθετων εργασιών σε υποεργασίες, β) ανάθεση των υποεργασιών στους διαθέσιμους πράκτορες μέσω διαπραγμάτευσης με τρόπο ανάλογο με τις απλές εργασίες και γ) ανασύνθεση των σύνθετων εργασιών που αναλαμβάνει κάθε πράκτορας.

Αναφορικά με τις σύνθετες εργασίες αναλυμένες σε υποεργασίες, πρέπει να σημειωθεί ότι δεν είναι όλα τα υποσύνολά τους *αποδεκτά* (valid). Μη αποδεκτό είναι το υποσύνολο υποεργασιών, που περιλαμβάνει δευτερεύουσες υποεργασίες και δεν περιέχει την αντίστοιχη κύρια. Για να διαπιστωθεί αν ένα τυχαίο υποσύνολο υποεργασιών είναι αποδεκτό, ελέγχεται για κάθε υποεργασία του αν είναι κύρια ή αν είναι δευτερεύουσα και η αντίστοιχη κύρια περιέχεται επίσης στο υποσύνολο. Ένα μη αποδεκτό υποσύνολο μπορεί να μετατραπεί στον αντίστοιχο αποδεκτό, αν αφαιρεθεί η δευτερεύουσα υποεργασία που δημιουργεί το πρόβλημα και προστεθεί η αντίστοιχη κύρια.

Σενάριο 2.1α.[19]

> Έστω οι εργασίες {$f_0(3)$, $f_1(1)$}. Οι υποεργασίες είναι 0-f0, 1-f΄0, 2-f΄0, 3-f1. Οι 0-f0 και 3-f1 είναι κύριες, οι 1-f΄0, 2-f΄0 δευτερεύουσες.
>
> Αποδεκτά είναι τα υποσύνολα {}, {f0}, {f0, f'0}, {f0, f'0, f'0}, {f1}, {f0, f1}, {f0, f'0, f1}, {f0, f'0, f'0, f1}. Μη αποδεκτό είναι το {f'0}, του οποίου το αντίστοιχο αποδεκτό είναι {f'0} ∪ {f0} – {f'0} = {f0}. Επίσης, το {f'0, f'0} που μετατρέπεται στο {f'0, f'0} ∪ {f0} – {f'0} = {f0, f'0}. Ακόμη, το {f'0, f'0, f'0} που μετατρέπεται στο {f'0, f'0, f'0} ∪ {f0} – {f'0} = {f0, f'0, f'0}. Τέλος, το {f'0, f'0, f'0, f1} που μετατρέπεται στο {f'0, f'0, f'0, f1} ∪ {f0} – {f'0} = {f0, f'0, f'0, f1}. □

Μετά τις παραπάνω διευκρινήσεις, θα ξαναδούμε την εφαρμογή του ΜΠΣ σε τομείς απλών και σύνθετων εργασιών και θα εστιάσουμε σε κάποιες λεπτομέρειες.

---

[19] Το σενάριο 2.1 αναφέρεται σε σύνθετες εργασίες και παρουσιάζεται σταδιακά. Περιλαμβάνει τα υποσενάρια 2.1α, 2.1β, κοκ.

2.2. Λεπτομέρειες πρωτοκόλλου και στρατηγικών.

Αλγόριθμος 2.1: ΜΠΣ

Προαιρετικός offline υπολογισμός με χρήση της κατάλληλης μεθόδου Επανάληψη μέχρι να υπάρξει συμφωνία ή αποτυχία { Παραγωγή επόμενης πρότασης με χρήση της κατάλληλης μεθόδου Αποστολή στον αντίπαλο Παραλαβή πρότασης αντιπάλου Αποτίμηση προτάσεων }

Παρατηρούμε ότι για τις προτάσεις που γίνονται κατά τη διαπραγμάτευση, πρέπει να ισχύουν οι παρακάτω σχέσεις:

- κάθε επόμενη πρόταση πρέπει να είναι τουλάχιστο τόσο καλή για τον αντίπαλο όσο και η προηγούμενη, δηλαδή στο γύρο t ο πράκτορας A_M προτείνει στον αντίπαλο A_O $d_{t,M}$ τέτοιο ώστε
$$u_O(d_{t-1,M}) \leq u_O(d_{t,M}) \text{ ή } c_O(d_{t-1,M}) \geq c_O(d_{t,M}), \text{ για } t>1.$$
- κάθε πρόταση πρέπει να μην είναι χειρότερη για τον ίδιο τον πράκτορα από την αρχική αλληλεπίδραση, δηλαδή ο πράκτορας A_M προτείνει $d_{t,M}$ τέτοιο ώστε
$$u_M(d_{t,M}) \geq 0 \text{ ή } c_M(e) \geq c_M(d_{t,M}), \text{ για } t\geq 1.$$

Έστω totalcost το άθροισμα του κόστους όλων των εργασιών, δηλαδή
$$\text{totalCost} = \sum c(t_i), \forall t_i \in T.$$

Ειδικά για τις απλές εργασίες κάνουμε την παρατήρηση ότι, λαμβάνοντας υπόψη το γεγονός ότι τα D_M και D_O μιας πρότασης δεν περιέχουν επικαλύψεις, μπορεί εύκολα να υπολογιστεί το κόστος μιας πρότασης για τον αντίπαλο, αν είναι γνωστό το κόστος για τον πράκτορα. Ειδικότερα ισχύει ότι
$$c_O(d_M) = \text{totalCost} - c_M(d_M). \text{ [σχέση 2.1]}$$
Επομένως ισχύει $u_M(d) = c_M(e) - c_M(d) = c_M(e) - \text{totalcost} + c_O(d)$. Όμως, $u_O(d) = c_O(e) - c_O(d) \Rightarrow c_O(d) = c_O(e) - u_O(d)$. Άρα,
$$u_M(d) = c_M(e) + c_O(e) - \text{totalcost} - u_O(d).$$
Αρχικό σημείο για τον πράκτορα A_M είναι αυτό στο οποίο
$$u_O(d_{1,M}) = 0 \text{ ή } c_O(d_M) = c_O(e).$$
Τελικό σημείο είναι αυτό στο οποίο
$$c_M(d_M) = c_M(e).$$
Αποδεκτές προτάσεις για διαπραγμάτευση είναι αυτές μεταξύ αρχικού και τελικού σημείου, δηλαδή
$$c_O(d_M) \leq c_O(e) \text{ και } c_M(d_M) \leq c_M(e). \text{ [σχέση 2.2]}$$

Ειδικά για τις απλές εργασίες, λαμβάνοντας υπόψη τη σχέση 2.1, θα έχουμε για το αρχικό σημείο

$$c_M(d_M) \leq totalCost - c_O(e).$$

Συνολικά για τις αποδεκτές προτάσεις η σχέση 2.2 διαμορφώνεται σε

$$totalCost - c_O(e) \leq c_M(d_M) \leq c_M(e).$$

Αναφορικά με τις ισοδύναμες προτάσεις, δηλαδή αυτές που έχουν ίδιο κόστος, ενδιαφέρει ουσιαστικά μόνο μία κάθε φορά. Κι αυτό γιατί σε κάθε γύρο ο πράκτορας επιτρέπεται να προτείνει κάτι με μεγαλύτερη ή ίση χρησιμότητα για τον αντίπαλο. Το να κάνει διαφορετική πρόταση με την ίδια χρησιμότητα, δηλαδή ισοδύναμη, δεν έχει κανένα νόημα, είναι ίδιο με το να παραμείνει στη θέση του. Μπορούμε λοιπόν να μην το επιτρέψουμε για να επιταχύνουμε τη διαδικασία.

Ειδικά για τις απλές εργασίες, αν μια πρόταση είναι ισοδύναμη για τον ένα, είναι και για τον αντίπαλο. Με άλλα λόγια, όταν δυο προτάσεις έχουν την ίδια χρησιμότητα ή κόστος για ένα πράκτορα A_M, έχουν το ίδιο κόστος μεταξύ τους και για τον αντίπαλο A_O. Αυτό ισχύει γιατί αν $c_M(d) = c_M(d') \Leftrightarrow totalCost - c_O(d) = totalCost - c_O(d') \Leftrightarrow c_O(d) = c_O(d')$.

Πρέπει να τονιστεί τέλος ότι σε μια πρόταση $d = <D_0,D_1>$ τα D_0 και D_1 είναι *συμπληρωματικά*, δηλαδή δεν περιέχουν επικαλύψεις και το $D_0 \cup D_1$ είναι πάντα το ίδιο και ίσο με $T_0 \cup T_1$. Αυτό δεν ισχύει για τα T_0 και T_1 που παρουσιάζουν επικαλύψεις και γι' άλλωστε έχει νόημα η διαδικασία της διαπραγμάτευσης.

Στο προηγούμενο κεφάλαιο τέθηκαν κάποια ζητήματα ως προς το ΜΠΣ και ο τρόπος αντιμετώπισή τους που προτείνεται στη βιβλιογραφία. Στη συνέχεια θα επεκταθούμε περισσότερο σε αυτά, προτείνοντας και κάποιες εναλλακτικές λύσεις, όπου είναι δυνατό.

Πρώτη πρόταση διαπραγμάτευσης.

Η πρώτη πρόταση κάθε πράκτορα πρέπει να είναι τουλάχιστο τόσο καλή για τον αντίπαλο όσο και η αρχική αλληλεπίδραση, γιατί διαφορετικά ο αντίπαλος θα την απορρίψει οπωσδήποτε. Από τις υπόλοιπες διαθέσιμες προτάσεις, μπορεί να διαλέξει αυτή που προτιμά περισσότερο. Δηλαδή συνολικά ο πράκτορας A_M πρέπει να προτείνει d_M τέτοιο ώστε

$$u_O(d_M) \geq 0 \text{ ή } c_O(e) \geq c_O(d_M) \text{ και } \max(u_M(d_M)) \text{ ή } \min(c_M(d_M)).$$

Πολιτικές υποχώρησης.

Σε κάθε γύρο ένας πράκτορας μπορεί είτε να υποχωρήσει, χάνοντας σε χρησιμότητα, είτε να παραμείνει στη θέση του, ρισκάροντας σύγκρουση. Όσο περισσότερο ρισκάρει, τόσο πιο *ριψοκίνδυνος* θεωρείται, ενώ στην αντίθετη περίπτωση χαρακτηρίζεται *συντηρητικός*. Στην πράξη, θα μπορούσαν να χρησιμοποιηθούν διάφορες *πολιτικές υποχώρησης (policies)*, όπως:

- Σε κάθε γύρο να υποχωρούν και οι δύο. Στην περίπτωση αυτή ο τερματισμός θα γίνει πιο γρήγορα, αλλά υπάρχει κίνδυνος να υποχωρήσει κάποιος περισσότερο από ότι χρειάζεται. Επιπλέον, αν κάποιος γνωρίζει ότι ο αντίπαλος ακολουθεί αυτή την πολιτική, μπορεί να τον εκμεταλλευτεί και να μην υποχωρεί καθόλου ο ίδιος. Συνεπώς, η πολιτική είναι μη σταθερή.
- Σε κάθε γύρο να επιλέγει κανείς με τυχαίο τρόπο αν θα υποχωρήσει ή όχι. Είναι όμως πολύ πιθανό να επιλέξουν και οι δυο να μην υποχωρήσουν και να επέλθει σύγκρουση, ενώ το σύνολο διαπραγμάτευσης περιλαμβάνει προτάσεις συμφέρουσες και για τους δυο. Συνεπώς, η πολιτική είναι μη αποτελεσματική.
- Να επιλέγει το σύστημα με τυχαίο τρόπο ποιος από τους δυο θα υποχωρήσει. Εναλλακτικά, αν κανείς έχει υποχωρήσει πολλές φορές στο παρελθόν, να μειώνεται η πιθανότητα να υποχωρήσει ξανά.
- Να υποχωρούν με τη σειρά, μια ο ένας, μια ο άλλος.
- Να υποχωρεί αυτός που έχει τη μικρότερη τιμή του υπολογιζόμενου ρίσκου με βάση τη στρατηγική Zeuthen που περιγράφηκε στο κεφάλαιο 1.

Τελευταίος γύρος διαπραγμάτευσης.

Στην περίπτωση που στον τελευταίο γύρο διαπραγμάτευσης και οι δυο πράκτορες αποδεχτούν την πρόταση του αντίπαλου, ποια από τις δυο θα επικρατήσει τελικά; Υπάρχουν διάφορες ιδέες που μπορούν να χρησιμοποιηθούν κατά περίπτωση:

- Να γίνει τυχαία επιλογή μιας από τις δύο προτάσεις, σύμφωνα με την επεκτεταμένη στρατηγική Zeuthen.
- Να επιλεγεί η πρόταση που μεγιστοποιεί το γινόμενο των χρησιμοτήτων των δυο πρακτόρων. Αν υπάρχουν περισσότερες από μια τέτοιες προτάσεις να επιλεγεί αυτή που μεγιστοποιεί το άθροισμα των χρησιμοτήτων των δυο πρακτόρων (κριτήριο δικαιοσύνης).
  $\max\{u_M(d_M)*u_O(d_M), u_M(d_O)*u_O(d_O)\}$ ή $\max\{u_M(d_M)+u_O(d_M), u_M(d_O)+u_O(d_O)\}$

- Να επιλεγεί η πρόταση που ελαχιστοποιεί τη διαφορά των χρησιμοτήτων (κριτήριο δικαιοσύνης).

$$\min\{u_M(d_M)-u_O(d_M),\ u_M(d_O)-u_O(d_O)\}$$

- Να επιλεγεί η πρόταση που αφήνει τον ένα πράκτορα όσο το δυνατό πιο ικανοποιημένο, αρκεί (προαιρετικά) να μην απομείνει ο αντίπαλος με μηδενική χρησιμότητα (κριτήριο μέγιστης ικανοποίησης ενός).

Ελάχιστη ικανή υποχώρηση.

Σημειώνουμε ότι αν η υποχώρηση δεν είναι ικανή, τότε θα πρέπει στον επόμενο γύρο να ξαναϋποχωρήσει ο ίδιος πράκτορας. Αν από την άλλη πλευρά ο πράκτορας υποχωρήσει πάρα πολύ, τότε χάνει σε χρησιμότητα χωρίς λόγο. Μια λύση θα ήταν να υποθέσει ότι θα κάνει στον επόμενο γύρο την ελάχιστη *δυνατή* υποχώρηση και με βάση αυτή να υπολογίσει την τιμή του ρίσκου και για τους δυο πράκτορες. Αν η ισορροπία του ρίσκου αλλάξει, τότε αυτή είναι η ελάχιστη *ικανή* υποχώρηση. Διαφορετικά, δοκιμάζει την αμέσως επόμενη ελάχιστη *δυνατή* υποχώρηση, κοκ. Με τον τρόπο αυτό, αυξάνει ελαφρά το υπολογιστικό κόστος, εφόσον το ρίσκο υπολογίζεται δυο φορές, μία δοκιμαστικά και μία κανονικά, μειώνεται όμως το επικοινωνιακό κόστος. Σε πραγματικές συνθήκες μπορεί να υπολογίσει ο προγραμματιστής τι από τα δυο συμφέρει και ανάλογα να ρυθμίσει τον πράκτορα.

Εύρεση επόμενης καλύτερης πρότασης.

Τέθηκε στο προηγούμενο κεφάλαιο το ερώτημα της εύρεσης επόμενης καλύτερης πρότασης, που πληροί τις προϋποθέσεις του πρωτοκόλλου και είναι η πιο συμφέρουσα για τον πράκτορα. Το πρόβλημα αυτό είναι ιδιαίτερα πολύπλοκο, μπορεί να απαιτήσει υπολογισμούς της τάξης του $2^{|T|}$.

Για την επίλυση αυτού του σοβαρού ζητήματος, παρατηρούμε την ομοιότητα που υπάρχει με τα γνωστά προβλήματα του αθροίσματος υποσυνόλου (subset sum) και του σακιδίου (knapsack). Το *subset sum*[20] πρόβλημα είναι το εξής: δοθέντος ενός συνόλου ακεραίων, υπάρχει υποσύνολο με άθροισμα ακριβώς W; (πρόβλημα απόφασης). Και κατ' επέκταση ποιο υποσύνολο έχει άθροισμα το πολύ W και πιο είναι αυτό το μέγιστο άθροισμα; (πρόβλημα βελτιστοποίησης). Το *knapsack*[21] πρόβλημα είναι το εξής: δοθέντος ενός συνόλου αντικειμένων που

[20] [Cormen 2001 κεφ.35.5], [Wikipedia]
[21] [Wikipedia], [Black 2005]

χαρακτηρίζονται από βάρος και τιμή, ποιο είναι το υποσύνολο αντικειμένων με μέγιστο άθροισμα τιμής, έτσι ώστε το άθροισμα βάρους να μην ξεπερνά το W;

Επιστρέφοντας στους ΤΠΕ, κάθε πράκτορας A_M αναζητά την καταλληλότερη από τις υποψήφιες προτάσεις του συνόλου διαπραγμάτευσης $d=<D_M,D_O>$[22] με κόστος $c_M(d)$ για τον πράκτορα και $c_O(d)$ για τον αντίπαλο. Λαμβάνοντας υπόψη ότι το D_O είναι συμπληρωματικό του D_M ουσιαστικά αναζητείται μόνο το κατάλληλο D_M. Έτσι, κάθε πράκτορας αναζητά σε κάθε γύρο ένα υποσύνολο εργασιών D_M τέτοιο ώστε να έχει ελάχιστο δυνατό κόστος για τον ίδιο (ή μέγιστο για τον αντίπαλο), αλλά ταυτόχρονα κόστος αντιπάλου μικρότερο από το κόστος της προηγούμενης πρότασης. Η πρώτη συνθήκη τον συμφέρει, η δεύτερη επιβάλλεται από το ΜΠΣ.

Η συσχέτιση αυτού του προβλήματος με τα γνωστά subset sum και knapsack είναι προφανής. Επομένως, χρησιμοποιούμε τις λύσεις που έχουν προταθεί κατά καιρούς[23] για τα δυο προβλήματα και τις προσαρμόζουμε στις απαιτήσεις μας, για να ορίσουμε κάποιες μεθόδους εύρεσης επόμενης πρότασης.

Μια πρώτη προσέγγιση είναι κάθε πράκτορας να υπολογίζει με τη σειρά τις καταλληλότερες προτάσεις, έτσι ώστε για κάθε νέο υπολογισμό να ξεκινάει από την αρχή. Αν $W = c_M(e)$ το τελικό σημείο, για κάθε βήμα $w = 1,\ldots W$, υπολογίζει ένα υποσύνολο εργασιών με κόστος το πολύ w. Είναι δυνατό είτε να υπολογίζεται το νέο υποσύνολο σε κάθε γύρο της διαπραγμάτευσης, είτε να υπολογίζεται μια λίστα με όλα τα υποσύνολα πριν ξεκινήσει η διαπραγμάτευση, δηλαδή offline. Η μέθοδος αυτή είναι εκθετική $O(W^2 2^N)$.

Μια βελτίωση του παραπάνω είναι να υπολογίζεται η λίστα με τα υποσύνολα των εργασιών με τρόπο που να γίνεται εκμετάλλευση της πληροφορίας που υπολογίστηκε στα προηγούμενα βήματα. Έστω ότι στο βήμα x-1, περιλαμβάνονται στη λίστα όλα τα υποσύνολα που περιέχουν τις εργασίες $t_0, \ldots, t_{x-1}$. Στο βήμα x, προστίθενται στη λίστα τα προηγούμενα υποσύνολα επαυξημένα με την εργασία t_x. Κατά τα άλλα ο υπολογισμός γίνεται offline και μέχρι το όριο W, όπως πριν. Ο αλγόριθμος είναι στη χειρότερη περίπτωση εκθετικός, απλούστερος όμως από τον προηγούμενο. Παρακάτω θα αναφερθούμε αναλυτικά στην «*Εκθετική μέθοδο ακρίβειας*».

Μια παραλλαγή του παραπάνω, που δίνει προσεγγιστικά αποτελέσματα, είναι σε κάθε βήμα από τη λίστα υποσυνόλων να 'κόβονται' τα υποσύνολα που έχουν μεταξύ τους κοντινό άθροισμα κόστους με βάση κάποιο παράγοντα δ. Παρακάτω θα αναφερθούμε αναλυτικά στην «*Προσεγγιστική μέθοδο ψαλιδίσματος*».

---

[22] ή $<D_O,D_M>$ ανάλογα με τις τιμές των Μ και Ο. Κατά σύμβαση είναι πάντα $<D_0,D_1>$.
[23] [Cormen 2001], [Martello 1990], [Wikipedia]

Λύσεις μπορούν να προκύψουν με τη χρήση τεχνικών δυναμικού προγραμματισμού και απληστίας, όπως θα συζητηθεί παρακάτω. Στην πρώτη περίπτωση προκύπτει η «*Ψευδοπολυωνυμική μέθοδος ακρίβειας*».

Μια άλλη ιδέα είναι να μην γίνεται κανένας offline υπολογισμός και σε κάθε γύρο ο πράκτορας να αναλαμβάνει μια από τις εργασίες του αντιπάλου. Η επιλογή μπορεί να γίνει με κάποιο κριτήριο. Παρακάτω θα αναφερθούμε αναλυτικά στην «*Ευρετική μέθοδο*».

Οι δυο μέθοδοι ακρίβειας που προτείνονται είναι βασισμένες στη θεωρία παιγνίων και δίνουν βέλτιστες λύσεις. Είναι όμως χρονοβόρες και δαπανηρές σε χώρο. Αντίθετα, οι άλλες δυο δίνουν προσεγγιστικές λύσεις που είναι υπο-βέλτιστες. Όμως προκύπτουν γρηγορότερα, τόσο η πρόταση κάθε γύρου, όσο και ο τερματισμός της διαπραγμάτευσης.

2.3. Εκθετική μέθοδος ακρίβειας.

Με την εκθετική μέθοδο ακρίβειας (EMA), πριν ξεκινήσει η διαπραγμάτευση, κάθε πράκτορας υπολογίζει, offline και αξιοποιώντας προηγούμενα αποτελέσματα, όλα τα πιθανά υποσύνολα εργασιών που αντιστοιχούν στο σύνολο διαπραγμάτευσης και το κόστος τους.

Έστω DL μια λίστα υποσυνόλων εργασιών $\{D_0, D_1, \ldots\}$ κόστους $C_{DL} = \{C_0, C_1, \ldots\}$. Ορίζουμε $DL \cup t_x = \{D_0 \cup t_x, D_1 \cup t_x, \ldots\}$ με κόστος $C_{DL \cup tx} = C_{DL} + c_x = \{C_0 + c_x, C_1 + c_x, \ldots\}$. Η mergeSubsets(DL, $DL \cup t_x$) συγχωνεύει και ταξινομεί τις δυο λίστες και δημιουργεί νέα λίστα DL. Με βάση τα παραπάνω, μπορεί να βρεθεί το δυναμοσύνολο των εργασιών P(T) και το αντίστοιχο κόστος με μια διαδικασία N βημάτων:

Αρχικά DL = {∅}.
Σε κάθε βήμα x = 0…N-1,
  DL = mergeSubsets(DL, $DL \cup t_x$).

Αναλυτικά, έχουμε:

Αρχικά	DL = {∅}
x = 0	DL = {∅,{t_0}}
x = 1	DL = {∅,{t_0},{t_1},{t_0, t_1}}
x = 2	DL = {∅,{t_0},{t_1},{t_0, t_1},{t_2},{t_0, t_2},{t_1, t_2},{t_0, t_1, t_2}}
…	
x = N-1	DL = {∅,{t_0},{t_1},{t_0, t_1},…,{t_0, t_1, t_2, …, t_{N-1}}} = P(T)

Στο επόμενο στάδιο, μετά τη δημιουργία του δυναμοσυνόλου, διαγράφουμε όσα υποσύνολα εργασιών δεν είναι pareto βέλτιστα και ατομικά ορθολογικά. Έτσι, απορρίπτουμε όσα έχουν κόστος μικρότερο του αρχικού σημείου και μεγαλύτερο του τελικού, ενώ από τα ισοδύναμα υποσύνολα κρατάμε μόνο ένα κάθε φορά. Τέλος, επιθυμούμε τα υποσύνολα να είναι ταξινομημένα κατά φθίνουσα σειρά κόστους για τον αντίπαλο. Αυτό για τις απλές εργασίες ισοδυναμεί με αύξουσα σειρά κόστους για τον πράκτορα. Οι παραπάνω λειτουργίες πραγματοποιούνται με την trimFinalList(DL).

Σημειώνεται ότι η απαλοιφή υποσυνόλων με κόστος μεγαλύτερο του τελικού σημείου και η ταξινόμησή τους, μπορεί, αντί να γίνεται μια φορά στο τέλος, να ενσωματωθεί στη mergeSubsets και να πραγματοποιείται σε κάθε βήμα της διαδικασίας. Έτσι, περιορίζεται το πλήθος των στοιχείων της λίστας και η διαδικασία γίνεται πιο γρήγορη και αποδοτική.

Στη συνέχεια θα δούμε ένα σενάριο offline υπολογισμού για απλές εργασίες.

Σενάριο 2.2α.[24]

Έστω οι πράκτορες a και b και έστω οι εργασίες $\{t_0, t_1, t_2, t_3\}$ με κόστος {1, 2, 3, 4}. Αρχική αλληλεπίδραση $e = <\{t_0, t_1, t_2, t_3\}, \{t_2, t_3\}>$.

Ισχύει $cost_a(e) = 10$ και $cost_b(e) = 7$. Στον πράκτορα b έχουμε:

βήμα 0. $DL = \{\varnothing\}$, $C = \{0\}$

βήμα 1. $DL = mergeSubsets(DL, DL \cup \{t_0\}) = \{\varnothing, \{t_0\}\}$
$C = mergeSubsets(C, C+c_0) = \{0, 1\}$

βήμα 2. $DL = mergeSubsets(DL, DL \cup \{t_1\}) = \{\varnothing, \{t_0\}, \{t_1\}, \{t_0,t_1\}\}$
$C = mergeSubsets(C, C+c_1) = \{0, 1, 2, 3\}$

βήμα 3. $DL = \{\varnothing, \{t_0\}, \{t_1\}, \{t_0,t_1\}, \{t_2\}, \{t_0,t_2\}, \{t_1,t_2\}, \{t_0,t_1,t_2\}\}$
$C = mergeSubsets(C, C+c_2) = \{0, 1, 2, 3, 3, 4, 5, 6\}$

βήμα 4.

$DL \cup \{t_3\} = \{\{t_3\},\{t_0,t_3\},\{t_1,t_3\},\{t_0,t_1,t_3\},\{t_2,t_3\},\{t_0,t_2,t_3\},\{t_1,t_2,t_3\},\{t_0,t_1,t_2,t_3\}\}$

$C+\{c_3\} = \{4, 5, 6, 7, 7, 8, 9, 10\}$

$DL = mergeSubsets(\{\varnothing, \{t_0\}, \{t_1\}, \{t_0,t_1\}, \{t_2\}, \{t_0,t_2\}, \{t_1,t_2\}, \{t_0,t_1,t_2\}\},$
$\{\{t_3\}, \{t_0,t_3\}, \{t_1,t_3\}, \{t_0,t_1,t_3\}, \{t_2,t_3\}, \{t_0,t_2,t_3\}, \{t_1,t_2,t_3\}, \{t_0,t_1,t_2,t_3\}\}) = \{\varnothing,$
$\{t_0\},\{t_1\},\{t_0,t_1\},\{t_2\},\{t_0,t_2\},\{t_3\},\{t_1,t_2\},\{t_0,t_3\},\{t_0,t_1,t_2\},\{t_1,t_3\},\{t_0,t_1,t_3\},\{t_2,t_3\}\}$

$C = mergeSubsets(\{0, 1, 2, 3, 3, 4, 5, 6\}, \{4, 5, 6, 7, 7, 8, 9, 10\}) =$
$= \{0, 1, 2, 3, 3, 4, 4, 5, 5, 6, 6, 7, 7\}$

Τελικά, $DL = trimFinalList(DL) =$
$= \{\varnothing,\{t_0\},\{t_1\},\{t_0,t_1\},\{t_0,t_2\},\{t_1,t_2\},\{t_0,t_1,t_2\},\{t_0,t_1,t_3\}\}$
και $C = trimFinalList(C) = \{0, 1, 2, 3, 4, 5, 6, 7\}$ □

[24] Το σενάριο 2.2 αναφέρεται σε απλές εργασίες και παρουσιάζεται σταδιακά. Περιλαμβάνει τα υποσενάρια 2.2α, 2.2β, κοκ.

Ο αλγόριθμος 2.2 που ακολουθεί περιλαμβάνει τη βασική ιδέα του offline υπολογισμού για απλές εργασίες με χρήση της εκθετικής μεθόδου ακρίβειας.

Αλγόριθμος 2.2: Offline υπολογισμός για την EMA & απλές εργασίες

```
calculateNS_method1_simple {
    DL = {∅}, C = {0}
    for (x=0 to N-1) {
        DL' = {}, C' = {}
        for (i=0 to |DL|-1) {
            DL' = DL' ∪ {DL_i ∪ {t_x}}
            C' = C' ∪ {C_i + c_x}
        }
        mergeSubsets(DL,DL',C,C')
    }
    trimFinalList(DL,C)
}
mergeSubsets(DL,DL',C,C') {
    DL" = {}, C" = {}
    m1 = 0, m2 = 0
    while (m1 < |DL|-1 and m2 < |DL'|-1)
        if (C_m1 < C'_m2) {
            if (C_m1 <= lastPoint)
                copySubset(m1,DL,DL",C,C")
            m1 = m1 + 1
        }
        else {
            if (C'_m2 <= lastPoint)
                copySubset(m2,DL',DL",C',C")
            m2 = m2 + 1
        }
    while (m1 < |DL|-1) {
        if (C_m1 <= lastPoint)
            copySubset(m1,DL,DL",C,C")
        m1 = m1 + 1
    }
    while (m2 < |DL'|-1) {
        if (C'_m2 <= lastPoint)
            copySubset(m2,DL',DL",C',C")
        m2 = m2 + 1
    }
    DL = DL", C = C"
}
trimFinalList(DL,C) {
    for (i=0 to |DL|-1) {
        if (C_i < initialPoint or C_i = C_i-1)
            deleteSubset(i,DL,C)
    }
}
```

Η copySubset(m, DL, DL1, C, C1) αντιγράφει το στοιχείο m της λίστας DL, το DL_m, στην DL1. Ομοίως το C_m στη C1. Η deleteSubset(m, DL, C) διαγράφει τα DL_m, C_m από τις λίστες DL και C.

Η διαδικασία σε N βήματα της δημιουργίας των DL' = $DL \cup t_x$ και της συγχώνευσης των DL και DL' απαιτεί στη χειρότερη περίπτωση πλήθος πράξεων $\sum_{x=0}^{N-1}(\sum_{i=0}^{|DL|-1} c)$, όπου $|DL|=2^x$, δηλαδή είναι $\sum_{x=0}^{N-1} c2^x$ και άρα $O(2^N)$. Η trimFinalList είναι επίσης στη χειρότερη περίπτωση πολυπλοκότητας $O(2^N)$. Αυτή είναι και η συνολική πολυπλοκότητα της μεθόδου EMA για απλές εργασίες.

Ειδικά για τις σύνθετες εργασίες, η διαδικασία είναι παρόμοια με τις απλές με κάποιες όμως ουσιαστικές διαφοροποιήσεις, στις λεπτομέρειες που αφορούν την ύπαρξη εξαρτήσεων μεταξύ των υποεργασιών, τις προϋποθέσεις χαρακτηρισμού μιας πρότασης ως pareto βέλτιστης, τον καθορισμό του αρχικού σημείου και τις ιδιότητες των ισοδύναμων προτάσεων.

Αρχικά οι Q σύνθετες εργασίες μετατρέπονται σε N υποεργασίες, για τις οποίες ξεκινά η διαδικασία υπολογισμού των υποσυνόλων και του αντίστοιχου κόστους. Οι διαφορές με την περίπτωση των απλών εργασιών αναλύονται στη συνέχεια.

Α. Κατά τον υπολογισμό σε N βήματα της λίστας με υποσύνολα υποεργασιών και το αντίστοιχο κόστος πράκτορα και αντιπάλου, διαπιστώνουμε ότι:

Ι. Δεν είναι όλα τα υποσύνολα *αποδεκτά*. Στο υποκεφάλαιο 2.1 περιγράφηκαν τα αποδεκτά και μη υποσύνολα εργασιών και οι μετατροπές τους. Στην offline διαδικασία τα μη αποδεκτά υποσύνολα εργασιών διαγράφονται και δεν τυγχάνουν περεταίρω επεξεργασίας, γιατί η αντικατάστασή τους στη λίστα από τα αντίστοιχα αποδεκτά θα οδηγούσε στην εμφάνιση του ίδιου ακριβώς υποσυνόλου πολλές φορές.

ΙΙ. Το κόστος μιας πρότασης για τον αντίπαλο δεν προκύπτει αυτόματα από το κόστος πράκτορα, όπως στις απλές εργασίες. Έτσι πρέπει να υπολογίζεται και αυτό και να αποθηκεύεται κάπου. Ο υπολογισμός του κόστους αντιπάλου για κάθε πρόταση είναι μια διαδικασία που περιγράφεται στη συνέχεια. Σημειώνεται ότι συμβολίζουμε με A_M τον πράκτορα και με A_O τον αντίπαλο.

Έστω D_M ένα αποδεκτό υποσύνολο υποεργασιών. Το αντίστοιχο υποσύνολο για τον αντίπαλο είναι D_O που είναι το συμπληρωματικό του D_M ή αν είναι μη αποδεκτό, το αντίστοιχο αποδεκτό. Ισχύει $d = <D_M,D_O>$ ή $<D_O,D_M>$. Το κόστος πράκτορα είναι $c_M(D_M) = c_M(d)$ και το κόστος αντιπάλου $c_O(D_O) = c_O(d)$. Ισχύει $c_M(\varnothing) = 0$ και $c_O(\varnothing) = totalCost$.

Έστω η εργασία t_i με κόστος c_i. Υποθέτουμε ότι το νέο υποσύνολο $D'_M = D_M \cup \{t_i\}$ είναι αποδεκτό. Το νέο κόστος πράκτορα θα είναι $c_M(d') = c_M(d) + c_i$. Το αντίστοιχο υποσύνολο αντιπάλου θα είναι $D'_O = D_O - \{t_i\}$. Το νέο κόστος αντιπάλου θα είναι $c_O(d') = c_O(d) - c'_i$. Για την ποσότητα c'_i διακρίνουμε τις περιπτώσεις:

α) η t_i είναι κύρια, δηλαδή είναι η πρώτη υποεργασία της σύνθετης εργασίας που αφαιρείται από το D_O, και απομένουν δευτερεύουσες στο D_O. Το D'_O δεν είναι αποδεκτό στην παρούσα μορφή του. Τότε πρέπει να αφαιρεθεί από το κόστος αντιπάλου όχι το κόστος της κύριας υποεργασίας (που είναι το c_i), αλλά το κόστος μιας δευτερεύουσας, δηλαδή $c'_i = c_{slave(ti)}$.

β) η t_i είναι δευτερεύουσα και είναι η τελευταία υποεργασία της σύνθετης εργασίας που αφαιρείται από το D_O. Τότε πρέπει να αφαιρεθεί από το κόστος αντιπάλου όχι το κόστος της δευτερεύουσας υποεργασίας, αλλά το κόστος της κύριας, δηλαδή $c'_i = c_{master(ti)}$.

γ) στις υπόλοιπες περιπτώσεις, ισχύει $c'_i = c_i$.

Σενάριο 2.1β.

Έστω οι εργασίες $\{f_0(3), f_1(1)\}$ με $c_0 = 1+n$ και $c_1 = 1+n$. Οι υποεργασίες είναι 0-f0, 1-f΄0, 2-f΄0, 3-f1. Οι 0-f0 και 3-f1 είναι κύριες, οι 1-f΄0, 2-f΄0 δευτερεύουσες. Οι υποεργασίες έχουν κόστος c0=2, c'0=1, c'0=1, c1=2.

Αν $d = <\{\}\{f0,f'0,f'0,f1\}>$ και t_i = 0-f0, τότε υπάγεται στην περίπτωση (α), οπότε ισχύει $c_i = 2$ και $c'_i = c_{slave(ti)} = 1$. Τότε $d' = <\{f0\}\{f'0,f'0,f1\}>$, $cost_M(d') = cost_M(d) + c_i = 0 + 2 = 2$ και $cost_O(d') = cost_O(d) - c'_i = 6 - 1 = 5$.

Αν $d = <\{f0\}\{f'0,f'0,f1\}>$ και t_i = 1-f'0, τότε υπάγεται στην περίπτωση (γ), οπότε ισχύει $c_i = 1$ και $c'_i = c_i = 1$. Τότε $d' = <\{f0,f'0\}\{f'0,f1\}>$, $cost_M(d') = cost_M(d) + c_i = 2 + 1 = 3$ και $cost_O(d') = cost_O(d) - c'_i = 5 - 1 = 4$.

Αν $d = <\{f0,f'0\}\{f'0,f1\}>$ και t_i = 2-f'0, τότε υπάγεται στην περίπτωση (β), οπότε ισχύει $c_i = 1$ και $c'_i = c_{slave(ti)} = 2$. Τότε $d' = <\{f0,f'0,f'0\}\{f1\}>$ $cost_M(d') = cost_M(d) + c_i = 3 + 1 = 4$ και $cost_O(d') = cost_O(d) - c'_i = 4 - 2 = 2$.

Αν $d = <\{f0,f'0,f'0\}\{f1\}>$ και t_i = 3-f1, τότε υπάγεται στην περίπτωση (γ), οπότε ισχύει $c_i = 2$ και $c'_i = c_i = 2$. Τότε $d' = <\{f0,f'0,f'0,f1\}\{\}>$, $cost_M(d') = cost_M(d) + c_i = 4 + 2 = 6$ και $cost_O(d') = cost_O(d) - c'_i = 2 - 2 = 0$. □

III. Σύμφωνα με το ΜΠΣ, σε κάθε νέο γύρο της διαπραγμάτευσης πρέπει να γίνει πρόταση που θα έχει για τον αντίπαλο μεγαλύτερη ή ίση χρησιμότητα με πριν. Συνεπώς, διευκολύνει τα υποσύνολα που θα προκύψουν από τον offline υπολογισμό να είναι ταξινομημένα κατά φθίνουσα σειρά κόστους αντιπάλου. Αυτό δεν ισοδυναμεί με αύξουσα σειρά κόστους πράκτορα, όπως στις απλές εργασίες.

Β. Μετά την ολοκλήρωση των N βημάτων, έχει δημιουργηθεί η λίστα με τα αποδεκτά υποσύνολα υποεργασιών και το αντίστοιχο κόστος πράκτορα και αντιπάλου. Από τη λίστα πρέπει τώρα να διαγραφούν τα υποσύνολα με κόστος μικρότερο του αρχικού σημείου και αυτά που αντιστοιχούν σε ισοδύναμα για τον αντίπαλο μοιράσματα. Οι διαφορές με την περίπτωση των απλών εργασιών αναλύονται στη συνέχεια.

IV. Αναφορικά με το αρχικό σημείο, απαλείφονται τα υποσύνολα με κόστος αντιπάλου μεγαλύτερο του $c_O(e)$. Δεν ισχύει όπως στις απλές εργασίες η ισοδύναμη απαλοιφή υποσυνόλων με κόστος πράκτορα μικρότερο του totalCost – $c_O(e)$.

V. Σε περίπτωση που υπάρχουν προτάσεις ίσου κόστους για τον αντίπαλο, επιλέγεται αυτή που δίνει το μικρότερο κόστος στον πράκτορα

VI. Τέλος, με ένα ακόμη πέρασμα ελέγχουμε για κάθε υποψήφια πρόταση, μήπως ακολουθεί άλλη (καλύτερη για τον αντίπαλο) που είναι ίδια ή καλύτερη και για τον πράκτορα.

Σημειώνεται ότι οι έλεγχοι V και VI γίνονται για να εξασφαλιστεί ότι οι προτάσεις που απομένουν είναι pareto βέλτιστες.

Στη συνέχεια θα δούμε ένα σενάριο offline υπολογισμού για σύνθετες εργασίες.

Σενάριο 2.1γ.

Έστω οι εργασίες $\{f_0(3), f_1(1)\}$ με $c_0 = 1+n$ και $c_1 = 1+n$. Οι υποεργασίες είναι 0-f0, 1-f΄0, 2-f΄0, 3-f1. Οι 0-f0 και 3-f1 είναι κύριες, οι 1-f΄0, 2-f΄0 δευτερεύουσες. Οι υποεργασίες έχουν κόστος c0=2, c'0=1, c'0=1, c1=2. Αρχική αλληλεπίδραση $e = <\{f_0(3), f_1(1)\}, \{f_0(3), f_1(1)\}>$.

Ισχύει $cost_a(e) = 6$ και $cost_b(e) = 6$. Στον πράκτορα a έχουμε:

βήμα 0. $DL = \{\varnothing\}$, $C_M = \{0\}$, $C_O = \{6\}$

βήμα 1. $DL = mergeSubsets(DL, DL \cup \{f0\}) = \{\varnothing, \{f0\}\}$

$C_M = mergeSubsets(C_M, C_M + c_{f0}) = \{0, 2\}$

$C_O = mergeSubsets(C_O, C_O - c'_{f0}) = \{6, 5^*\}$

* $c'_{f0} = c_{slave(f0)} = c_{f'0} = 1$, βλέπε παράδειγμα 2.1β.

βήμα 2. $DL \cup \{f'0\}) = \{\{f0\}^{\#}, \{f0,f'0\}\} = \{\{f0,f'0\}\}$

$^{\#}$ απαλοιφή του μη αποδεκτού $\{f'0\}$, βλέπε παράδειγμα 2.1α.

$DL = \text{mergeSubsets}(DL, DL \cup \{f'0\}) = \{\varnothing, \{f0\}, \{f0,f'0\}\}$

$C_M = \text{mergeSubsets}(C_M, C_M + c_{f'0}) = \{0, 2, 3\}$

$C_O = \text{mergeSubsets}(C_O, C_O - c'_{f'0}) = \{6, 5, 4^{*}\}$

* $c'_{f'0} = c_{f'0} = 1$, βλέπε παράδειγμα 2.1β.

βήμα 3. $DL \cup \{f'0\} = \{\{f0\}^{\#}, \{f0,f'0\}, \{f0,f'0,f'0\}\} = \{\{f0,f'0\}, \{f0,f'0,f'0\}\}$

$^{\#}$ απαλοιφή του μη αποδεκτού $\{f'0\}$, βλέπε παράδειγμα 2.1α.

$C_M + c_{f'0} = \{2+1, 3+1\} = \{3, 4\}$

$C_O - c'_{f'0} = \{5-1^{*}, 4-2^{**}\} = \{4, 2\}$

* $c'_{f'0} = c_{f'0} = 1$, βλέπε παράδειγμα 2.1β.

** $c'_{f'0} = c_{master(f'0)} = c_{f0} = 2$, βλέπε παράδειγμα 2.1β.

Λαμβάνοντας υπόψη και την ταξινόμηση, έχουμε:

$DL = \{\varnothing, \{f0\}, \{f0,f'0\}, \{f0,f'0\}, \{f0,f'0,f'0\}\}$

$C_M = \text{mergeSubsets}(C_M, C_M + c_{f'0}) = \{0, 2, 3, 3, 4\}$

$C_O = \text{mergeSubsets}(C_O, C_O - c'_{f'0}) = \{6, 5, 4, 4, 2\}$

βήμα 4. $DL \cup \{f1\} = \{\{f1\}, \{f0,f1\}, \{f0,f'0,f1\}, \{f0,f'0,f1\}, \{f0,f'0,f'0,f1\}\}$

$C_M + c_{f1} = \{2, 4, 5, 5, 6\}$

$C_O - c'_{f1}{}^{*} = \{4, 3, 2, 2, 0\}$

* $c'_{f1} = c_{f1} = 2$, βλέπε παράδειγμα 2.1β.

Λαμβάνοντας υπόψη και την ταξινόμηση, έχουμε:

$DL = \{\varnothing, \{f0\}, \{f1\}, \{f0,f'0\}, \{f0,f'0\}, \{f0,f1\}, \{f0,f'0,f'0\},$
$\{f0,f'0,f1\}, \{f0,f'0,f1\}, \{f0,f'0,f'0,f1\}\}$

$C_M = \{0, 2, 2, 3, 3, 4, 4, 5, 5, 6\}$

$C_O = \{6, 5, 4, 4, 4, 3, 2, 2, 2, 0\}$

Σημειώνεται ότι για όλα τα παραπάνω, η τιμές κόστους πράκτορα είναι μικρότερες ή ίσες του τελικού σημείου που είναι 6.

Επίσης, για όλα τα παραπάνω, η τιμές κόστους του αντιπάλου είναι μεγαλύτερες ή ίσες του αρχικού σημείου που είναι 0.

Στη συνέχεια, από τις προτάσεις ίσου κόστους για τον αντίπαλο, επιλέγεται αυτή με το μικρότερο κόστος πράκτορα

$DL = \{\varnothing, \{f0\}, \{f1\}, \{f0,f1\}, \{f0,f'0,f'0\}, \{f0,f'0,f'0,f1\}\}$

$C_M = \{0, 2, 2, 4, 4, 6\}$, $C_O = \{6, 5, 4, 3, 2, 0\}$.

Τέλος, για κάθε υποψήφια πρόταση ελέγχουμε μήπως ακολουθεί άλλη (καλύτερη για τον αντίπαλο) που είναι ίδια ή καλύτερη για τον πράκτορα.

$DL = \{\varnothing, \{f1\}, \{f0,f'0,f'0\}, \{f0,f'0,f'0,f1\}\}$

$C_M = \{0, 2, 4, 6\}$, $C_O = \{6, 4, 2, 0\}$. □

Ο αλγόριθμος 2.3 που ακολουθεί περιλαμβάνει τη βασική ιδέα του offline υπολογισμού για σύνθετες εργασίες με χρήση της εκθετικής μεθόδου ακρίβειας.

Αλγόριθμος 2.3: Offline υπολογισμός για την EMA & σύνθετες εργασίες

```
calculateNS_method1_complex {
    DL = {∅}, C_M = {0}, C_O = {totalCost}
    for (x=0 to N-1) {
        DL' = {}, C'_M = {}, C'_O = {}
        for (i=0 to |DL|-1)
            if (isValid(DL_i ∪ {t_x})) {
                DL' = DL' ∪ {DL_i ∪ {t_x}}
                C'_M = C'_M ∪ {C_M(i) + c_x}
                c'_x = calculateSubtaskCost(x, DL_x)
                C'_O = C'_O ∪ {C_O(i) – c'_x}
            }
            mergeSubsets(DL,DL',C_M,C'_M,C_O,C'_O)
    }
    trimFinalList(DL,C_M,C_O)
}
trimFinalList(DL,C_M,C_O)
    for (i=0 to |DL|-1) {                                          *
        if (C_O(i) < initialPoint)
            deleteSubset(i,DL,C_M,C_O)
        else if (C_O(i) = C_O(i-1) and C_M(i) < C_M(i-1))
            deleteSubset(i-1,DL,C_M,C_O)
    }
    DL' = {}, C'_M = {}, C'_O = {}
    for (i=0 to |DL|-1) {                                          **
        for (k=i+1 to |DL|-1)
            if (C_M(k) < C_M(i))
                i = k
        copySubset(i,DL,DL',C_M,C'_M,C_O,C'_O)
    }
}
isValid(subset∪{t_x}) {
    return (t_x is master) or (t_x is slave and t_k master of t_x and t_k ∈ subset)
}
calculateSubtaskCost(x, subset) {
    if ((t_x is master) and (∃ t_j: t_j slave of t_x and t_j ∉ subset))
        return c_j
    else if ((t_x is slave) and (t_k master of t_x)
        and not (∃ t_j: t_j slave of t_k and t_j ∈ subset))
        return c_k
    else
        return c_x
}
```

Η mergeSubsets είναι παρόμοια με τις απλές εργασίες, μόνο που ταξινομεί τις τρεις λίστες κατά φθίνουσα σειρά κόστους. Σημειώνεται ότι οι δυο ανακυκλώσεις της trimFinalList προσπαθούν να εξασφαλίσουν ότι οι προτάσεις που απομένουν θα είναι pareto βέλτιστες, επειδή δεν θα υπάρχει περίπτωση να αυξηθεί η χρησιμότητα για τον ένα πράκτορα, χωρίς να μειωθεί η χρησιμότητα του άλλου.

Η calculateSubtaskCost(x, subset) είναι $O(m_x)$ όπου m_x το πλήθος των υποεργασιών της εργασίας t_x. Συνολικά η calculateSubtaskCost είναι στη χειρότερη περίπτωση $O(\max\{m_x\}) < O(N)$.

Αντίστοιχα προς τον αλγόριθμο για απλές εργασίες, η διαδικασία των N βημάτων της δημιουργίας των $DL' = DL \cup t_x$ και της συγχώνευσης των DL και DL' απαιτεί στη χειρότερη περίπτωση πλήθος πράξεων $\Sigma_{x=0}^{N-1}$ ($\Sigma_{i=0}^{|DL|-1}$ c $O(\max\{m_x\})$), όπου $|DL|=2^x$, δηλαδή είναι $\Sigma_{x=0}^{N-1}$ $c2^x O(\max\{m_x\})$ και άρα $O(\max\{m_x\}2^N) < O(N2^N)$.

Αναφορικά με την trimFinalList, η πρώτη ανακύκλωση * είναι $O(K)$ με $K \leq 2^N$ και η δεύτερη ανακύκλωση ** είναι $O(L^2)$ με $L \leq 2^N$ και πιθανώς L πολύ μικρότερο του 2^N, αν έχουν γίνει πολλές διαγραφές από τη λίστα. Δηλαδή η πολυπλοκότητα της trimFinalList είναι $O(K+L^2) \leq O(2^{2N})$ και πιθανώς $<< O(2^{2N})$.

Συνολικά, η πολυπλοκότητα της EMA για σύνθετες εργασίες είναι $O(\max\{m_x\}2^N+K+L^2)$ που είναι στη χειρότερη περίπτωση $O(2^{2N})$.

Έχοντας υπολογίσει offline την λίστα υποσυνόλων υποεργασιών που αντιστοιχεί στο σύνολο διαπραγμάτευσης, μπορούμε να δούμε τη διαδικασία της διαπραγμάτευσης που είναι πλέον πολύ εύκολη. Στο ΜΠΣ κάθε πράκτορας προτείνει σε κάθε γύρο το επόμενο υποσύνολο της ταξινομημένης λίστας, που είναι καλύτερο για τον αντίπαλο και χειρότερο για τον ίδιο. Για απλές εργασίες, ο πράκτορας προτείνει να αναλάβει ο ίδιος τις υποεργασίες του υποσυνόλου D_i και ο αντίπαλος τις υποεργασίες του συμπληρωματικού υποσυνόλου. Το κόστος για καθένα είναι αντίστοιχα $C_{M(i)}$ και totalCost – $C_{M(i)}$. Για σύνθετες εργασίες, ο πράκτορας προτείνει να αναλάβει ο ίδιος τις υποεργασίες του υποσυνόλου D_i και ο αντίπαλος τις υποεργασίες του συμπληρωματικού υποσυνόλου, εφόσον είναι αποδεκτό. Διαφορετικά το μετατρέπει στο αντίστοιχο αποδεκτό. Το κόστος για καθένα είναι αντίστοιχα $C_{M(i)}$ και $C_{O(i)}$.

Αναφορικά με την ανάκτηση επόμενης πρότασης από τη λίστα, ισχύει αλγόριθμος 2.4 που είναι κοινός για απλές και σύνθετες εργασίες.

Αλγόριθμος 2.4: Ανάκτηση επόμενης πρότασης για την ΕΜΑ

```
Αρχικά
currentPos = 0

Σε κάθε γύρο
    D_M = D(currentPos)
    D_O = getComplementary(D_M)
    currentPos = currentPos+1
```

Η πολυπλοκότητα σε κάθε γύρο της διαπραγμάτευσης εξαρτάται από την εσωτερική αναπαράσταση της πληροφορίας. Όπως θα δούμε στο επόμενο κεφάλαιο, μπορεί να είναι O(N). Συνολικά υπάρχουν λιγότεροι από $\mathbf{2^N}$ γύροι.

2.4. Προσεγγιστική μέθοδος ψαλιδίσματος.

Η προσεγγιστική μέθοδος ψαλιδίσματος (ΠΜΨ) είναι παρόμοια με την εκθετική μέθοδο ακρίβειας, με την έννοια ότι χρησιμοποιούνται ταξινομημένες λίστες υποσυνόλων εργασιών που δημιουργούνται με μια διαδικασία N βημάτων. Η διαφορά είναι ότι στη mergeSubsets 'κόβονται' (trim) από τη λίστα τα υποσύνολα που έχουν μεταξύ τους κοντινό άθροισμα κόστους με βάση κάποιο παράγοντα ψαλιδίσματος δ. Ειδικότερα, αν για τις εργασίες t_z και t_y ισχύει $c_y/(1+\delta) \leq c_z \leq c_y$, τότε η t_y διαγράφεται από τη λίστα και θα αντιπροσωπεύεται πλέον από την t_z. Από το χρήστη δίδεται ένας παράγοντας προσέγγισης ε, που δηλώνει σε πιο βαθμό επιθυμεί το στοιχείο που κόβεται από τη λίστα να προσεγγίζεται από τον αντιπρόσωπο, π.χ. κατά 40%. Θα ισχύει $\delta = \varepsilon/2N$.

Στη συνέχεια θα δούμε ένα σενάριο offline υπολογισμού για απλές εργασίες.

Σενάριο 2.2β.

Έστω οι εργασίες $\{t_0, t_1, t_2, t_3\}$ με κόστος {1, 2, 3, 4}. Αρχική αλληλεπίδραση $e = <\{t_0, t_1, t_2, t_3\}, \{t_2, t_3\}>$. Ισχύει $cost_a(e) = 10$, $cost_b(e) = 7$.

Από το σενάριο που αντιμετωπίζει το ίδιο παράδειγμα με την ΕΜΑ έχουμε για τον πράκτορα b:

βήμα 0. DL = {∅}, C = {0}

βήμα 1. DL = {∅, $\{t_0\}$}, C = {0, 1}

βήμα 2. DL = {∅, $\{t_0\}$, $\{t_1\}$, $\{t_0,t_1\}$}, C = {0, 1, 2, 3}

βήμα 3. DL = {∅, $\{t_0\}$, $\{t_1\}$, $\{t_0,t_1\}$, $\{t_2\}$, $\{t_0,t_2\}$, $\{t_1,t_2\}$, $\{t_0,t_1,t_2\}$}
C = {0, 1, 2, 3, 3, 4, 5, 6}

βήμα 4. DL = {∅, $\{t_0\}$, $\{t_1\}$, $\{t_0,t_1\}$, $\{t_2\}$, $\{t_0,t_2\}$, $\{t_3\}$, $\{t_1,t_2\}$, $\{t_0,t_3\}$, $\{t_0,t_1,t_2\}$}, $\{t_1,t_3\}$, $\{t_0,t_1,t_3\}$, $\{t_2,t_3\}$}
C = {0, 1, 2, 3, 3, 4, 4, 5, 5, 6, 6, 7, 7}

Αν $\varepsilon = 0.1 \Rightarrow \delta = 0.1/8 = 0.0125$, τότε έχουμε:

βήμα 0, 1 και 2, ως έχουν.

βήμα 3. $DL = \{\varnothing, \{t_0\}, \{t_1\}, \{t_0,t_1\}, \{t_0,t_2\}, \{t_1,t_2\}, \{t_0,t_1,t_2\}\}$
$C = \{0, 1, 2, 3, 4, 5, 6\}$

βήμα 4. $DL = \{\varnothing, \{t_0\}, \{t_1\}, \{t_0,t_1\}, \{t_0,t_2\}, \{t_1,t_2\}, \{t_0,t_1,t_2\}, \{t_0,t_1,t_3\}\}$
$C = \{0, 1, 2, 3, 4, 5, 6, 7\}$

Σημειώνεται ότι όλα τα παραπάνω είναι μεγαλύτερα του αρχικού σημείου. □

Ο αλγόριθμος 2.5 offline υπολογισμού για απλές εργασίες είναι παρόμοιος με πριν. Διαφοροποιείται μόνο η mergeSubsets στα υπογραμμισμένα σημεία.

Αλγόριθμος 2.5: Διαφορές σε offline υπολογισμό για ΠΜΨ & απλές εργασίες

```
mergeSubsets(DL,DL',C,C') {
   DL" = {}, C" = {}
   m1 = 0, m2 = 0, old_m = 0
   while (m1 < |DL|-1 and m2 < |DL'|-1)
      if (C_m1 < C'_m2) {
         if (C_m1 <= lastPoint) {                                    *
            copySubset(m1,DL,DL",C,C"); old_m = m1
         }
         m1 = m1 + 1
      }
      else {
         if (C'_m2 <= lastPoint and not approximate_found(m2,old_m)) {   *
            copySubset(m2,DL',DL",C',C"); old_m = m2
         }
         m2 = m2 + 1
      }
   while (m1 < |DL|-1) {
      if (C_m1 <= lastPoint) {                                       *
         copySubset(m1,DL,DL",C,C"); old_m = m1
      }
      m1 = m1 + 1
   }
   while (m2 < |DL'|-1) {
      if (C'_m2 <= lastPoint and not approximate_found(m2,old_m)) {   *
         copySubset(m2,DL',DL",C',C"); old_m = m2
      }
      m2 = m2 + 1
   }
   DL = DL", C = C"
}
boolean approximate_found(m, old_m) {
   if (m == 0) return false
   if (C(m) <= firstPoint) return false                               #
   if (C(old_m) >= C(m)/(1+δ))% return true
}
```

* Υπενθυμίζεται ότι σε κάθε βήμα i συγχωνεύονται και ταξινομούνται οι δυο λίστες DL και DL$\cup\{t_i\}$. Η πρώτη έχει δείκτη m1 και η δεύτερη m2. Από την λίστα DL έχουν ήδη 'κοπεί' τα υποσύνολα παραπλήσιου κόστους από τον έλεγχο * του προηγούμενου βήματος. Έτσι στο βήμα αυτό, ο έλεγχος * γίνεται μόνο στη λίστα DL$\cup\{t_i\}$.

Προσοχή: Δεν θέλουμε να 'κοπούν' στοιχεία που αντιπροσωπεύονται από τιμές μικρότερες του αρχικού σημείου, γιατί αυτές στη συνέχεια θα διαγραφούν.

% Ο έλεγχος C(old_m) <= C(m) δεν χρειάζεται καν να γίνει, γιατί η λίστα είναι ταξινομημένη.

Στη συνέχεια θα δούμε ένα σενάριο offline υπολογισμού για σύνθετες εργασίες.

Σενάριο 2.1δ.

Έστω οι εργασίες $\{f_0(3), f_1(1)\}$ με $c_0 = 1+n$ και $c_1 = 1+n$. Οι υποεργασίες είναι 0-f0, 1-f΄0, 2-f΄0, 3-f1. Οι 0-f0 και 3-f1 είναι κύριες, οι 1-f΄0, 2-f΄0 δευτερεύουσες. Οι υποεργασίες έχουν κόστος c0=2, c'0=1, c'0=1, c1=2. Αρχική αλληλεπίδραση $e = <\{f_0(3), f_1(1)\}, \{f_0(3), f_1(1)\}>$. Ισχύει $cost_a(e) = 6$ και $cost_b(e) = 6$.

Από το σενάριο που αντιμετωπίζει το ίδιο παράδειγμα με την EMA έχουμε για τον πράκτορα a:

βήμα 0. $DL = \{\varnothing\}$, $C_M = \{0\}$, $C_O = \{6\}$

βήμα 1. $DL = \{\varnothing, \{f0\}\}$, $C_M = \{0, 2\}$, $C_O = \{6, 5\}$

βήμα 2. $DL = \{\varnothing, \{f0\}, \{f0,f'0\}\}$, $C_M = \{0, 2, 3\}$, $C_O = \{6, 5, 4\}$

βήμα 3. $DL = \{\varnothing, \{f0\}, \{f0,f'0\}, \{f0,f'0\}, \{f0,f'0,f'0\}\}$

$C_M = \{0, 2, 3, 3, 4\}$

$C_O = \{6, 5, 4, 4, 2\}$

βήμα 4. $DL = \{\varnothing, \{f0\}, \{f1\}, \{f0,f'0\}, \{f0,f'0\}, \{f0,f1\},\{f0,f'0,f'0\},$
$\{f0,f'0,f1\},\{f0,f'0,f1\}, \{f0,f'0,f'0,f1\}\}$

$C_M = \{0, 2, 2, 3, 3, 4, 4, 5, 5, 6\}$

$C_O = \{6, 5, 4, 4, 4, 3, 2, 2, 2, 0\}$

Σημειώνεται ότι για όλα τα παραπάνω, η τιμές κόστους πράκτορα είναι μικρότερες ή ίσες του τελικού σημείου που είναι 6.

Αν $\varepsilon = 0.1 \Rightarrow \delta = 0.1/8 = 0.0125$, τότε έχουμε αντί για τα προηγούμενα:

βήμα 0, 1 και 2, ως έχουν.

βήμα 3. $DL = \{\varnothing, \{f0\}, \{f0,f'0\}, \{f0,f'0,f'0\}\}$

$C_M = \{0, 2, 3, 4\}$

C_O = {6, 5, 4, 2}

βήμα 4. DL = {∅, {f0}, {f0,f'0}, {f0,f1}, {f0,f'0,f'0}, {f0,f'0,f'0,f1}}

C_M = {0, 2, 3, 4, 4, 6}

C_O = {6, 5, 4, 3, 2, 0}

Για όλα τα παραπάνω, η τιμές κόστους του αντιπάλου είναι μεγαλύτερες ή ίσες του αρχικού σημείου που είναι 0.

Τέλος, για κάθε υποψήφια πρόταση ελέγχουμε μήπως ακολουθεί άλλη (καλύτερη για τον αντίπαλο) που είναι ίδια ή καλύτερη για τον πράκτορα.

DL = {∅, {f0}, {f0,f'0}, {f0,f'0,f'0}, {f0,f'0,f'0,f1}}

C_M = {0, 2, 3, 4, 6}

C_O = {6, 5, 4, 2, 0}. □

Ο αλγόριθμος offline υπολογισμού για σύνθετες εργασίες είναι παρόμοιος με πριν. Καταρχήν, διαφοροποιείται ελαφρώς η mergeSubsets. Επίσης, από την trimFinalList αφαιρείται εντελώς το παρακάτω τμήμα που επιλέγει από τις προτάσεις ίσου κόστους για τον αντίπαλο, αυτές με το μικρότερο κόστος πράκτορα. Το τμήμα αυτό είναι περιττό επειδή ούτως ή άλλως χάρη στο 'κόψιμο' της λίστας σε κάθε βήμα δεν υπάρχουν καθόλου προτάσεις ίσου κόστους για τον αντίπαλο.

```
else if (C_O(i) = C_O(i-1) and C_M(i) < C_M(i-1))
    deleteSubset(i-1,DL,C_M,C_O)
```

Επιπλέον, θα μπορούσε να αφαιρεθεί και το επόμενο τμήμα που ανατρέχει τη λίστα μια επιπλέον φορά και ελέγχει για κάθε υποψήφια πρόταση, μήπως ακολουθεί άλλη (καλύτερη για τον αντίπαλο) που είναι ίδια ή καλύτερη και για τον πράκτορα. Ο έλεγχος αυτός είναι χρονοβόρος και δεν είναι απαραίτητος, εφόσον η μέθοδος είναι απλώς προσεγγιστική.

```
for (i=0 to |DL|-1) {
    for (k=i+1 to |DL|-1)
        if (C_M(k) < C_M(i))
            i = k
    copySubset(i,DL,DL',C_M,C'_M,C_O,C'_O)
}
```

Τελικά η trimFinalList μπορεί να γίνει ως εξής:

```
trimFinalList2(DL,C_M,C_O)
for (i=0 to |DL|-1)
    if (C_O(i) < initialPoint)
        deleteSubset(i,DL,C_M,C_O)
}
```

Η διαδικασία ανάκτησης επόμενης πρότασης σε κάθε γύρο της διαπραγμάτευσης είναι ίδια με αυτή της εκθετικής μεθόδου ακρίβειας.

Από τα παραπάνω προκύπτει ότι με την ΠΜΨ η τελική λίστα υποσυνόλων εργασιών ή υποεργασιών δεν ισοδυναμεί με το σύνολο διαπραγμάτευσης. Υπενθυμίζεται ότι το σύνολο διαπραγμάτευσης περιέχει όλες τις προτάσεις που είναι pareto βέλτιστες και ατομικά ορθολογικές. Η τρέχουσα λίστα DL όμως καταρχήν μπορεί να μην περιέχει όλες τις απαραίτητες προτάσεις γιατί έχουν πραγματοποιηθεί ψαλιδίσματα. Δηλαδή η λίστα δεν είναι πλήρης. Επίσης, η λίστα μπορεί να περιέχει και άλλες προτάσεις που δεν είναι pareto βέλτιστες οι ίδιες, αποτελούν όμως καλές προσεγγίσεις άλλων βέλτιστων προτάσεων. Άλλωστε οι προτάσεις αυτές είναι σίγουρα καλύτερες από την αρχική αλληλεπίδραση και την πρόταση σύγκρουσης, οπότε συμφέρουν και τους δυο πράκτορες. Στο ζήτημα αυτό θα αναφερθούμε και παρακάτω, στο υποκεφάλαιο της αποτίμησης των μεθόδων.

Να σημειωθεί ότι για τις σύνθετες εργασίες έχουν γίνει περισσότερες απλοποιήσεις εις βάρος της ορθότητας του αποτελέσματος, οπότε μπορούμε να πούμε ότι γενικά η μέθοδος μάλλον δουλεύει καλύτερα για απλές εργασίες.

Από την άλλη πλευρά, η μέθοδος είναι γρηγορότερη από την EMA. Αποδεικνύεται[25] ότι η διαδικασία υπολογισμού της προσεγγιστικής λίστας υποσυνόλων και κόστους είναι πολυωνυμική ως προς τα N και 1/ε, ενώ η EMA ήταν εκθετική. Όσο για την trimFinalList2, έχει κανονικά πολυπλοκότητα χειρότερης περίπτωσης O(|DL|). Στην πράξη, χάρη στα διαδοχικά ψαλιδίσματα της λίστας το |DL| αποδεικνύεται πολύ μικρότερο του 2^N και τελικά η μέθοδος είναι συνολικά πολυωνυμική, τόσο για απλές όσο και για σύνθετες εργασίες.

2.5. Ψευδοπολυωνυμική μέθοδος ακρίβειας.

Η ψευδοπολυωνυμική μέθοδος ακρίβειας (ΨΜΑ) χρησιμοποιεί τεχνικές δυναμικού προγραμματισμού. Ο *δυναμικός προγραμματισμός*[26] - *ΔΠ* (dynamic programming) γενικά διασπά ένα αρχικό πρόβλημα σε απλούστερα υποπροβλήματα, τα οποία παρουσιάζουν σε κάποιο βαθμό επικαλύψεις. Τα υποπροβλήματα λύνονται με βέλτιστο τρόπο και η λύση τους αποθηκεύεται για

[25] [Cormen 2001 κεφ.35.5]
[26] [Cormen 2001 κεφ.15] , [Wikipedia]

να επαναχρησιμοποιηθεί. Ο κατάλληλος συνδυασμός των επιμέρους λύσεων δίνει την ολική βέλτιστη λύση.

Στην περίπτωσή μας, η διαδικασία offline υπολογισμού των υποσυνόλων που αντιστοιχούν στο σύνολο διαπραγμάτευσης και του κόστους τους μπορεί να πραγματοποιηθεί με τη χρήση δυναμικού προγραμματισμού. Το πρόβλημα της εύρεσης υποσυνόλου των N εργασιών κόστους το πολύ W = $c_M(e)$ που είναι το τελικό σημείο, μπορεί να διασπαστεί σε απλούστερα επικαλυπτόμενα υποπροβλήματα, των οποίων η λύση αποθηκεύεται σε πίνακες για να επαναχρησιμοποιηθεί.

Συγκεκριμένα, κάθε υποπρόβλημα αναζητά ένα υποσύνολο κάποιων από τις i πρώτες εργασίες, i = 0,…N-1, με κόστος μέγιστο αλλά το πολύ ίσο με w = 0, …, W-1 και το αποθηκεύει στο κελί subsets[j][w] του αντίστοιχου πίνακα. Ο δείκτης j θα εξηγηθεί στη συνέχεια. Στο κελί cM[j][w] αποθηκεύεται το αντίστοιχο κόστος. Σύμφωνα με τον αλγόριθμο, διακρίνουμε τις εξής περιπτώσεις:

α) Αν η εργασία i υπερβαίνει τον περιορισμό κόστους, δηλαδή αν c_i > w, δεν θα περιέχεται στο υποσύνολο και cM[j][w] = cM[j-1][w].

β) Διαφορετικά, αν το κόστος cM[j-1][w] χωρίς την εργασία i, είναι μεγαλύτερο από το κόστος cM[j-1][w-c_i] + c_i με την εργασία i, τότε δεν θα περιέχεται στο υποσύνολο και cM[j][w] = cM[j-1][w].

γ) Διαφορετικά, η εργασία i θα περιέχεται στο υποσύνολο και cM[j][w] = cM[j-1][w-c_i] + c_i.

Στο σημείο αυτό κάνουμε την παρατήρηση ότι επειδή ο αλγόριθμος χρειάζεται μια επιπλέον μηδενική πρώτη γραμμή και στήλη, το κελί [j][w] των πινάκων αντιστοιχεί κανονικά όχι στην εργασία j, αλλά στην j-1 (έχουμε μετατόπιση κατά ένα). Για λόγους καθαρότητας αλγορίθμου, χρησιμοποιούμε δυο διαφορετικούς δείκτες: για τις εργασίες i από 0 έως N-1 και για τους πίνακες j από 1 έως N. Στην πραγματικότητα ισχύει i=j-1. Το παραπάνω συμβαίνει γιατί η αρίθμηση των εργασιών ξεκινά από 0.

Ακολουθεί ο αλγόριθμος offline υπολογισμού της λίστας υποσυνόλων εργασιών με τη ΨΜΑ για απλές εργασίες. Η πολυπλοκότητα είναι Θ(NW), δηλαδή είναι ψευδοπολυωνυμική, γιατί το W μπορεί να εξαρτάται από το N.

Αλγόριθμος 2.6: Offline υπολογισμός για την ΨΜΑ & απλές εργασίες

```
for (w = 0 to W) {
    subsets[0][w] = {}
    cM[0][w] = 0                                          O(W)
}
for (i = 0 to N-1, j=1 to N) {
    cM[j][0] = 0                                          O(N)
    for (w = 1 to W) {                                    O(NW)
        withoutI = cM[j-1][w]
        withI = -1
        if (w >= c_i)
            withI = cM[j-1][w-c_i] + c_i
        if (withI > withoutI) {
            cM[j][w] = withI
            subsets[j][w] = subsets[j-1][w-c_i] ∪ {t_i}
        }
        else {
            cM[j][w] = withoutI
            subsets[j][w] = subsets[j-1][w]
        }
    }
}
for (currentPos = 0 to W) // find initial point
    if (cM[N][currentPos] >= totalCost – cost_O(T)        O(W)
        break
```

Στη συνέχεια θα δούμε ένα σενάριο offline υπολογισμού για απλές εργασίες.

Σενάριο 2.2γ.

Έστω οι εργασίες $\{t_0, t_1, t_2, t_3\}$ με κόστος $\{1, 2, 3, 4\}$. Αρχική αλληλεπίδραση $e = <\{t_0, t_1, t_2, t_3\}, \{t_2, t_3\}>$. Ισχύει $cost_a(e) = 10$, $cost_b(e) = 7$.

Για τον πράκτορα b, ισχύει $W = 7$.

Η πρώτη γραμμή των πινάκων είναι μηδενική. Για τα υπόλοιπα έχουμε:

- $cM[1][0] = 0$ και $subsets[1][0] = \{\}$
- $cM[1][1] = \max\{cM[0][1], cM[0][0]+1\} = 1$ (withI) και $subsets[1][1] = \{t_0\}$ …
- $cM[1][7] = \max\{cM[0][7], cM[0][6]+1\} = 1$ (withI) και $subsets[1][7] = \{t_0\}$
- $cM[2][1] = \max\{cM[1][1], -1\} = 1$ (withOutI) και $subsets[2][1] = \{t_0\}$
- $cM[2][2] = \max\{cM[1][2], cM[1][0]+2\} = 2$ (withI) και $subsets[2][1] = \{t_1\}$ …
- $cM[2][7] = \max\{cM[1][7], cM[1][5]+2\} = 3$ (withI) και $subsets[2][7] = \{t_0, t_1\}$ …
- $cM[3][7] = \max\{cM[2][7], cM[2][4]+3\} = 6$ (withI) και $subsets[3][7] = \{t_0, t_1, t_3\}$

  …
- $cM[4][6] = \max\{cM[3][6], cM[3][2]+4\} = 6$ (withOutI)

  και $subsets[4][6] = \{t_0, t_1, t_2\}$ …
- $cM[4][7] = \max\{cM[3][7], cM[3][3]+4\} = 7$ (withI) και $subsets[4][7] = \{t_0, t_1, t_3\}$

currentPos = 0

Πίνακας cM.

i \w	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	1	1	1	1	1	1
2	0	1	2	3	3	3	3	3
3	0	1	2	3	4	5	6	6
4	0	1	2	3	4	5	6	7

Πίνακας subsets.

i \w	0	1	2	3	4	5	6	7
0	{}	{}	{}	{}	{}	{}	{}	{}
1	{}	$\{t_0\}$	$\{t_0\}$	$\{t_0\}$	$\{t_0\}$	$\{t_0\}$	$\{t_0\}$	$\{t_0\}$
2	{}	$\{t_0\}$	$\{t_1\}$	$\{t_0,t_1\}$	$\{t_0,t_1\}$	$\{t_0,t_1\}$	$\{t_0,t_1\}$	$\{t_0,t_1\}$
3	{}	$\{t_0\}$	$\{t_1\}$	$\{t_0,t_1\}$	$\{t_0,t_2\}$	$\{t_1,t_2\}$	$\{t_0,t_1,t_2\}$	$\{t_0,t_1,t_3\}$
4	{}	$\{t_0\}$	$\{t_1\}$	$\{t_0,t_1\}$	$\{t_0,t_2\}$	$\{t_1,t_2\}$	$\{t_0,t_1,t_2\}$	$\{t_0,t_1,t_3\}$

Αναφορικά με τις σύνθετες εργασίες, η ΨΜΑ είναι αντίστοιχη της μεθόδου για τις απλές εργασίες, μόνο που υπολογίζει τα υποσύνολα και το κόστος υποεργασιών των σύνθετων εργασιών. Χρησιμοποιεί και πάλι τις αρχές του δυναμικού προγραμματισμού. Επειδή το κόστος αντιπάλου δεν προκύπτει άμεσα από το κόστος πράκτορα όπως στις απλές εργασίες, ο αλγόριθμος χρησιμοποιεί εκτός από τους πίνακες subsets και cM, ένα πίνακα cO, όπου αποθηκεύεται το αντίστοιχο κόστος αντιπάλου.

Αναφορικά με τον υπολογισμό του κόστους, αρχικά το κόστος πράκτορα που περιέχεται στο cM είναι 0 και το κόστος αντιπάλου του cO ισούται με totalCost. Για να υπολογίσουμε τα υπόλοιπα κελιά του πίνακα διακρίνουμε τις περιπτώσεις:

α) Αν $w \geq c_i$ και το τρέχον υποσύνολο με την υποεργασία i είναι αποδεκτό, δηλαδή είτε η i είναι κύρια, είτε η κύριά της περιλαμβάνεται στο υποσύνολο. Τότε subsets[j][w] = subsets[j-1][w-c_i] $\cup$ {t_i}, cM[j][w] = cM[j-1][w-c_i] + ci και cO[j][w] = cO[j-1][w-c_i] – c'_i, όπου c'_i η ίδια ποσότητα που ορίστηκε στη μέθοδο 1 και υπολογίζεται με τη calculateSubtaskCost(i, subsets[j-1][w-c_i]).

β) Διαφορετικά, μπορούμε να δοκιμάσουμε μήπως συμφέρει να προσθέσουμε στο subsets όχι μόνο τη συγκεκριμένη υποεργασία, αλλά μαζί και την κύριά της και όσες υποεργασίες της ίδιας ομάδας προηγούνται της i, δηλαδή όλες τις υποεργασίες από master(i) μέχρι i. Το κόστος όλων αυτών των υποεργασιών αθροιστικά θα είναι c_{ii}. Επίσης, c'_{ii} είναι το άθροισμα των c'_k = calculateSubtaskCost(k,subsets[j-1][w-c_{ii}] για τις παραπάνω εργασίες. Επομένως subsets[j][w] = subsets[j-1][w-c_{ii}] $\cup$ {$t_{master(i)}$, …, t_i}, cM[j][w] = cM[j-1][w-c_{ii}] + c_{ii} και cO[j][w] = cO[j-1][w-c_{ii}] – c'_{ii}.

γ) Διαφορετικά δεν συμφέρει να προστεθεί η υποεργασία i, ούτε μόνη της ούτε με τις προηγούμενες υποεργασίες της ίδιας εργασίας. Τότε έχουμε cM[j][w] = cM[j-1][w], cO[j][w] = cO[j-1][w] και subsets[j][w] = subsets[j-1][w].

Αλγόριθμος 2.7: Offline υπολογισμός για την ΨΜΑ & σύνθετες εργασίες.

```
for (w = 0 to W) {                                                    O(W)
    subsets[0][w] = {}
    cM[0][w] = 0
    cO[0][w] = totalCost
}
for (j = 1 to N, i = 0 to N-1) {
    cM[j][0] = 0                                                      O(N)
    cO[j][0] = totalCost
    if (slave(i))
        for (tk = tmaster(i) to ti) {                                 O(mN) *
            tii = tii ∪ {tk}
            cii = cii + ck
        }
    for (w = 1 to W) {
        cMwithoutI = cM[j-1][w]                                       O(NW)
        cOwithoutI = cO[j-1][w]
        swithoutI = subsets[j-1][w]
        cMwithI = -1
        cOwithI = 1000
        if (w >= ci and isValid(subsets[j-1][w-ci])∪{ti}) {
            cMwithI = cM[j-1][w-ci] + ci
            c'i = calculateSubtaskCost(i,subsets[j-1][w-ci]) **       O(mNW) *
            cOwithI = cO[j-1][w-ci] - c'i
            swithI = subsets[j-1][w-ci] ∪ {ti}
        }
        else if (slave(i) and w >= cii) {
            for (tk = tmaster(i) to ti)
              c'ii = c'ii + calculateSubtaskCost(k,subsets[j-1][w-cii])  O(m²NW) *
            cMwithI = cM[j-1][w-cii] + cii
            cOwithI = cO[j-1][w-cii] – c'ii
            swithI = subsets[j-1][w-cii] ∪ tii
        }
        if ((cOwithoutI < cOwithI) ||
        (cOwithI == cOwithoutI and cMwithoutI >= cMwithI)) {          O(NW)
            cM[j][w] = cMwithoutI
            cO[j][w] = cOwithoutI
            subsets[j][w] = swithoutI
        }
        else {                                                        O(NW)
            cM[j][w] = cMwithI
            cO[j][w] = cOwithI
            subsets[j][w] = swithI
        }
    }
}
for (currentPos = 0 to W) // find start point                         O(W)
    if (cO[N][currentPos] <= finalPoint)
        break
```

* m=max{m_x}, δηλαδή το μέγιστο πλήθος υποεργασιών των εργασιών.

** Η calculateSubtaskCost(x,subset) είναι ακριβώς ίδια με την EMA.

Η πολυπλοκότητα της μεθόδου για σύνθετες εργασίες είναι $O(m^2NW) << O(N^3W)$, δηλαδή και αυτή ψευδοπολυωνυμική.

Στη συνέχεια θα δούμε ένα σενάριο για σύνθετες εργασίες.

Σενάριο 2.1ε.

Έστω οι εργασίες {$f_0(3)$, $f_1(1)$} με $c_0 = 1+n$ και $c_1 = 1+n$. Υποεργασίες είναι {f0, f0, f0, f1} ή {0, 1, 2, 3}. Αρχική αλληλεπίδραση e = <{$f_0(3)$, $f_1(1)$} {$f_0(3)$, $f_1(1)$}>. Ισχύει $cost_a(e) = 6$ και $cost_b(e) = 6$. Επίσης, W = 6. Η συμπλήρωση των πινάκων γίνεται όπως παρακάτω.

- cO[0][w] = 6, cM[0][w] = 0 και subsets[0][w] = {}
- cO[1][0] = 6, cM[1][0] = 0 και subsets[1][0] = {} …

Γραμμή 4, εργασία i=3. c_i=2, c_i'=calculateSubTaskCost(3,subsets[3][2-2])=2.

- cO[4][1] = min{cO[3][1],1000} = min{6,1000} = 6 (withOutI)
  cM[4][1] = cM[3][1] = 0
  subsets[4][1] = subsets[3][1] = {}
- cO[4][2] = min{cO[3][2], cO[3][2–c_i] – c_i'} = min{5,4} = 4 (withI)
  cM[4][2] = cM[3][0] + c_i = 0 + 2 = 2
  subsets[4][2] = subsets[3][0] ∪ {f1} = {f1}
- cO[4][3] = min{cO[3][3], cO[3][3–c_i] – c_i'} = min{4,4} = 4
  cM[4][3] = max{cM[3][3], cM[3][3–c_i]+c_i} = max{3,2} = 3 (withOutI)
  subsets[4][3] = subsets[3][3] = {f0,f'0} ……
- cO[4][6] = min{cO[3][6],cO[3][6–c_i] – c_i'} = min{2,0} = 0 (withI)
  cM[4][6] = cM[3][6–c_i] + c_i = 4 + 2 = 6
  subsets[4][6] = subsets[3][6-ci] ∪ {f1} = {f0,f'0,f'0,f1}

cO	0	1	2	3	4	5	6
0	6	6	6	6	6	6	6
1	6	6	5	5	5	5	5
2	6	6	5	4	4	4	4
3	6	6	5	4	2	2	2
4	6	6	4	4	2	2	0

cM	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	0	2	2	2	2	2
2	0	0	2	3	3	3	3
3	0	0	2	3	4	4	4
4	0	0	2	3	4	5	6

subsets	1	2	3	4	5	6	7
0	{}	{}	{}	{}	{}	{}	{}
1	{}	{}	{f0}	{f0}	{f0}	{f0}	{f0}
2	{}	{}	{f0}	{f0,f'0}	{f0,f'0}	{f0,f'0}	{f0,f'0}
3	{}	{}	{f0}	{f0,f'0}	{f0,f'0,f'0}	{f0,f'0,f'0}	{f0,f'0,f'0}
4	{}	{}	{f1}	{f0,f'0}	{f0,f'0,f'0}	{f0,f'0,f1}	{f0,f'0,f'0,f1}

Η διαδικασία ανάκτησης επόμενης πρότασης σε κάθε γύρο της διαπραγμάτευσης είναι αντίστοιχη αυτή της EMA. Το τρέχον προτεινόμενο υποσύνολο αποθηκεύεται στο subsets[N][currentPos].

Η πολυπλοκότητα σε κάθε γύρο της διαπραγμάτευσης εξαρτάται από την εσωτερική αναπαράσταση της πληροφορίας. Συνολικά υπάρχουν λιγότεροι από W γύροι.

2.6. Ευρετική μέθοδος.

Στην ευρετική μέθοδο (EM) ο πράκτορας αρχικά αναθέτει όλες τις εργασίες στον αντίπαλο και σε κάθε βήμα της διαπραγμάτευσης επιλέγει μια εργασία του αντιπάλου και την αναλαμβάνει. Η εργασία μπορεί να επιλέγεται τυχαία, να είναι επόμενη στη σειρά, ή να προκύπτει με κάποιο άλλο κριτήριο, για παράδειγμα αυτή με το μικρότερο ή το μεγαλύτερο κόστος. Θα προτιμήσουμε να επιλεγεί η εργασία μικρότερου κόστους, δεδομένου ότι αυτό συμφέρει τον πράκτορα. Με βάση τα παραπάνω δεν απαιτείται κάποιος offline υπολογισμός, εκτός από μια προαιρετική αρχική ταξινόμηση των εργασιών.

Ακολουθεί ο αλγόριθμος για απλές εργασίες για τον πρώτο και για τους επόμενους γύρους της διαπραγμάτευσης.

Αλγόριθμος 2.8α: Εύρεση επόμενης πρότασης για την EM & απλές εργασίες

Πρώτος γύρος διαπραγμάτευσης

```
D_M = {}, C_M = 0
D_O = {t_0, ..., t_N-1}, C_O = totalCost
while (C_M < initialPoint) {          #
    t_i = getNextTask()
    D_M = D_M ∪ {t_i}
    C_M = C_M + c_i
    D_O = D_O – {t_i}
    C_O = C_O – c_i
    if (C_M > finalPoint)
        return null
}
return <D_M,D_O>
```

Αλγόριθμος 2.8β: Εύρεση επόμενης πρότασης για την ΕΜ & απλές εργασίες

Επόμενοι γύροι διαπραγμάτευσης

```
t_i = getNextTask()
D_M = D_M ∪ {t_i}
C_M = C_M + c_i
D_O = D_O − {t_i}
C_O = C_O − c_i
if (C_M > finalPoint)
    return null
return <D_M,D_O>
```

Έχουμε ≤N γύρους διαπραγμάτευσης.

Μετά τον πρώτο γύρο, η πολυπλοκότητα καθορίζεται από τη getNextTask() και είναι στη χειρότερη περίπτωση O(N) για μη ταξινομημένες εργασίες. Ειδικά για τον πρώτο γύρο, μπορεί να έχουμε μεγαλύτερη πολυπλοκότητα κατά τον προσδιορισμό του αρχικού σημείου (γραμμή #), αλλά μειώνονται αντίστοιχα οι γύροι διαπραγμάτευσης. Δηλαδή χωρίς ταξινόμηση έχουμε στη χειρότερη περίπτωση N γύρους x O(N), συνολικά πολυπλοκότητα $O(N^2)$.

Στην περίπτωση που έχουμε ταξινόμηση, απαιτείται μια offline διαδικασία με πολυπλοκότητα O(NlogN). Σε κάθε γύρο η getNextTask() θα ήταν πλέον O(1). Δηλαδή με ταξινόμηση έχουμε στη χειρότερη περίπτωση N γύρους x O(1) + O(NlogN), συνολικά πολυπλοκότητα O(NlogN).

Ακολουθεί ένα σενάριο εφαρμογής της μεθόδου σε απλές εργασίες.

Σενάριο 2.2δ.

Έστω οι εργασίες $\{t_0, t_1, t_2, t_3\}$ με κόστος {1, 2, 3, 4}. $e = <\{t_0, t_1, t_2, t_3\}, \{t_2, t_3\}>$. Ισχύει $cost_a(e) = 10$ και $cost_b(e) = 7$.

a –1 $D_M = \{\}$, $C_M = 0$ και $D_O = \{t_0, t_1, t_2, t_3\}$, $C_O = 10 > 7$, απορρίπτεται

$D_M = \{t_0\}$, $C_M = 1$ και $D_O = \{t_1, t_2, t_3\}$, $C_O = 9 > 7$, απορρίπτεται

$D_M = \{t_0,t_1\}$, $C_M = 3$ και $D_O = \{t_2, t_3\}$, $C_O = 7$, $<\{t_0,t_1\}\{t_2, t_3\}>$

b – 1 $D_M = \{\}$, $C_M = 0$ και $D_O = \{t_0, t_1, t_2, t_3\}$, $C_O = 10$, $<\{t_0, t_1, t_2, t_3\}\{\}>$

a – 2 $D_M = \{t_0,t_1,t_2\}$, $C_M = 6$ και $D_O = \{t_3\}$, $C_O = 4$, $<\{t_0,t_1,t_2\}\{t_3\}>$

b – 2 $D_M = \{t_0\}$, $C_M = 1$ και $D_O = \{t_1, t_2, t_3\}$, $C_O = 9$, $<\{t_1, t_2, t_3\}\{t_0\}>$

a – 3 $D_M = \{t_0,t_1,t_2,t_3\}$, $C_M = 10$ και $D_O = \{\}$, $C_O = 0$, $<\{t_0,t_1,t_2,t_3\}\{\}>$

b – 3 $D_M = \{t_0,t_1\}$, $C_M = 3$ και $D_O = \{t_2, t_3\}$, $C_O = 7$, $<\{t_2, t_3\}\{t_0,t_1\}>$ □

Η μέθοδος είναι παρόμοια και για σύνθετες εργασίες. Η διαφορά είναι ότι στη getNextTask() επιλέγεται η υποεργασία μικρότερου κόστους υπό την προϋπόθεση όμως ότι το παραγόμενο υποσύνολο υποεργασιών D_M να είναι αποδεκτό. Επίσης, αν το υποσύνολο του αντιπάλου δεν είναι αποδεκτό,

μετατρέπεται στο αντίστοιχο αποδεκτό. Τέλος, χρειάζεται προσοχή ο υπολογισμός του κόστους αντιπάλου.

Ακολουθεί ο αλγόριθμος για σύνθετες εργασίες για τον πρώτο γύρο διαπραγμάτευσης που είναι και πιο πολύπλοκος. Έχουν υπογραμμιστεί οι διαφορές από τις απλές εργασίες.

Αλγόριθμος 2.9: Εύρεση επόμενης πρότασης για την ΕΜ & σύνθετες εργασίες

Πρώτος γύρος διαπραγμάτευσης

```
D_M = {}, C_M = 0
D_O = {t_0, ..., t_N-1}, C_O = totalCost
while (C_O > initialPoint) {                          #
    t_i = getNextTask()                    O(N) ή O(1)
    D_M = D_M ∪ {t_i}
    C_M = C_M + c_i
    D_O = D_O − {t_i}
    D_O = makeValid(D_O)                       O(m)*
    c'_i = calculateSubtaskCost(i, D_M)        O(m)*
    C_O = C_O − c'_i
    if (C_M > finalPoint)
        return null
}
return <D_M,D_O>
```

* m=max{m_x}, δηλαδή το μέγιστο πλήθος υποεργασιών των εργασιών.

Η πολυπλοκότητα του αλγορίθμου είναι παρόμοια με πριν. Έχουμε ≤N γύρους διαπραγμάτευσης. Μετά τον πρώτο γύρο, η πολυπλοκότητα καθορίζεται από τη getNextTask() και είναι στη χειρότερη περίπτωση O(N) για μη ταξινομημένες εργασίες. Ειδικά για τον πρώτο γύρο, μπορεί να έχουμε μεγαλύτερη πολυπλοκότητα κατά τον προσδιορισμό του αρχικού σημείου (γραμμή #), αλλά μειώνονται αντίστοιχα οι γύροι διαπραγμάτευσης. Στην περίπτωση που έχουμε ταξινόμηση, απαιτείται μια offline διαδικασία με πολυπλοκότητα O(NlogN). Σε κάθε γύρο η getNextTask() θα ήταν πλέον O(m).

Μια λίγο πολυπλοκότερη παραλλαγή είναι στη getNextTask() να ελέγχεται αν υπάρχει και άλλη υποεργασία του αντιπάλου με ίσο κόστος που δίνει επίσης αποδεκτό υποσύνολο. Τότε από τις δυο επιλέγω αυτή που δίνει μικρότερο κόστος αντιπάλου. Η παραλλαγή αυτή δίνει ελαφρώς καλύτερα αποτελέσματα σε βάρος όμως της ταχύτητας. Για το λόγο αυτό υιοθετούμε την προηγούμενη έκδοση της μεθόδου.

Ακολουθεί ένα σενάριο εφαρμογής της μεθόδου σε σύνθετες εργασίες.

Σενάριο 2.1στ.

Έστω οι εργασίες $\{f_0(3), f_1(1)\}$ με $c_0 = 1+n$ και $c_1 = 1+n$. Αρχική αλληλεπίδραση $e = <\{f_0(3),f_1(1)\}\{f_0(3),f_1(1)\}>$. Υποεργασίες είναι {f0,f0,f0,f1} ή {0,1,2,3}. Ισχύει $cost_a(e) = 6$ και $cost_b(e) = 6$.

a – 1 $D_M = \{\}$, $C_M = 0$ και $D_O = \{t_0, t_1, t_2, t_3\}$, $C_O = 6$, <{}{f0,f'0,f'0,f1}>

b – 1 $D_M = \{\}$, $C_M = 0$ και $D_O = \{t_0, t_1, t_2, t_3\}$, $C_O = 6$, <{f0,f'0,f'0,f1}{}>

a – 2 $D_M = \{f0\}$, $C_M = 2$
και $D_O = \{f'0,f'0,f1\} = \{f0,f'0,f1\}$, $C_O = 5$, <{f0}{f0,f'0,f1}>

b – 2 $D_M = \{f0\}$, $C_M = 2$
και $D_O = \{f'0,f'0,f1\} = \{f0,f'0,f1\}$, $C_O = 5$, <{f0,f'0,f1}{f0}>

a – 3 $D_M = \{f0,f'0\}$, $C_M = 3$
και $D_O = \{f'0,f1\} = \{f0,f1\}$, $C_O = 4$, <{f0,f'0}{f0,f1}>

b – 3 $\{f0,f'0\}$, $C_M = 3$ και $D_O = \{f'0,f1\} = \{f0,f1\}$, $C_O = 4$, <{f0,f1}{f0,f'0}>

a – 4 $D_M = \{f0,f'0,f'0\}$, $C_M = 4$ και $D_O = \{f1\}$, $C_O = 2$, <{f0,f'0,f'0}{f1}>

b – 4 $D_M = \{f0,f'0,f'0\}$, $C_M = 4$ και $D_O = \{f1\}$, $C_O = 2$, <{f1}{f0,f'0,f'0}>

□

Κάνουμε την παρατήρηση ότι το αποτέλεσμα που προκύπτει με την ΕΜ δεν είναι απαραίτητα pareto βέλτιστο. Κι αυτό γιατί σε κάθε βήμα δεν ελέγχει όλες τις πιθανές προτάσεις μήπως και υπάρχει κάποια καλύτερη για τον ένα πράκτορα που δεν είναι χειρότερη για τον άλλο. Κάτι τέτοιο θα είχε δραματική επίδραση στην πολυπλοκότητα του αλγορίθμου, κι αυτό δεν έχει νόημα, γιατί όπως είπαμε επιθυμούμε η ΕΜ, όπως και ΠΜΨ, να δίνει αποτελέσματα υποβέλτιστα, αλλά γρήγορα. Στο ζήτημα αυτό θα αναφερθούμε και παρακάτω, στο υποκεφάλαιο της αποτίμησης των μεθόδων.

2.7. Αποτίμηση μεθόδων.

Οι μέθοδοι ΕΜΑ και ΨΜΑ, τόσο για απλές, όσο και για σύνθετες εργασίες, εντοπίζουν offline όλα τα στοιχεία του συνόλου διαπραγμάτευσης. Βασίζονται στη θεωρία παιγνίων και δίνουν βέλτιστα αποτελέσματα, που είναι τα ίδια και στις δυο περιπτώσεις. Είναι όμως ιδιαίτερα χρονοβόρες. Ως προς την εύρεση επόμενης πρότασης έχουν την ίδια πολυπλοκότητα. Ως προς το τμήμα του offline υπολογισμού, η ΕΜΑ είναι στη χειρότερη περίπτωση εκθετική, δηλαδή $O(2^N)$ για απλές και $O(2^{2N})$ για σύνθετες εργασίες και η ΨΜΑ ψευδοπολυωνυμική, δηλαδή O(NW) για απλές και $O(m^2NW)$ για σύνθετες

εργασίες. Το ποια από τις δύο υπερέχει εξαρτάται από τις τιμές των N και W, όπως φαίνεται και από τα επόμενα παραδείγματα.

Ι. Έστω N = 20 εργασίες και έστω ότι από την αρχική αλληλεπίδραση και από τις τιμές κόστους προκύπτει ότι W = 1000. Τότε η ΕΜΑ έχει πολυπλοκότητα της τάξης του 2^{20} και η ΨΜΑ από την άλλη 20x1000. Προφανώς υπερέχει κατά πολύ η ΨΜΑ.

ΙΙ. Έστω N = 4 εργασίες και έστω ότι από την αρχική αλληλεπίδραση και από τις τιμές κόστους προκύπτει ότι W = 10000000000. Τότε η ΕΜΑ έχει πολυπλοκότητα της τάξης του 2^4 και η ΨΜΑ από την άλλη 4x10000000000. Προφανώς υπερέχει κατά πολύ η ΕΜΑ.

Συνεπώς δεν υπάρχει μέθοδος άριστη για όλες τις περιπτώσεις. Μπορούμε, όμως, να πούμε ότι η ΨΜΑ υπερέχει της ΕΜΑ για μεγάλα N, δηλαδή όταν N > log(W) για απλές και N > log(m^2W)/2 για σύνθετες εργασίες.

Από την άλλη πλευρά, οι μέθοδοι ΠΜΨ και ΕΜ είναι πολύ γρηγορότερες από τις ΕΜΑ και ΨΜΑ, αλλά δε δίνουν πάντα το καλύτερο δυνατό αποτέλεσμα. Μάλιστα, δεν υπάρχει καμιά εγγύηση ότι το αποτέλεσμα θα είναι pareto βέλτιστο, επομένως μπορεί να μην ανήκει καν στο σύνολο διαπραγμάτευσης. Προτού όμως βιαστούμε να απορρίψουμε τις δυο μεθόδους πρέπει να λάβουμε υπόψη ότι η αρχική αλληλεπίδραση και η πρόταση σύγκρουσης κατά κανόνα δεν είναι επίσης pareto βέλτιστες και θα είναι χειρότερες από τη συμφωνία που θα προκύψει με μια προσεγγιστική μέθοδο.

Για παράδειγμα, έστω οι εργασίες $\{t_0,t_1,t_2,t_3,t_4\}$ κόστους {1000, 1000, 1000, 1000, 1000}. Αρχική αλληλεπίδραση e = $<\{t_0,t_1,t_2,t_3,t_4\}\{t_0,t_1,t_2,t_3,t_4\}>$ με $c_a(e)$=5000 και $c_b(e)$=5000. Έστω ότι μια μέθοδος ακρίβειας δίνει την πρόταση d = $<\{t_0,t_1,t_2\}\{t_3,t_4\}>$ με $c_a(d)$=3000 και $c_b(d)$=2000. Μια προσεγγιστική μέθοδος που δίνει γρήγορα τη μη pareto βέλτιστη πρόταση d΄ = $<\{t_0,t_1,t_2\}\{t_2,t_3,t_4\}>$ με $c_a(d')$=3000 και $c_b(d')$=3000 προφανώς δεν μπορεί να απορριφθεί αβασάνιστα.

Γενικά, οι μέθοδοι ΠΜΨ και ΕΜ μπορούν να εφαρμοστούν σε περιπτώσεις που δεν ενδιαφέρει η ορθότητα, αλλά η ταχύτητα της λύσης. Μια ιδέα για ειδικές περιπτώσεις θα ήταν η εφαρμογή της μεθόδου ΕΜ για μια άμεση συμβιβαστική λύση και, μόνο αν προκύψει σύγκρουση, εφαρμογή της ΕΜΑ ή της ΨΜΑ.

Τέλος, συγκρίνοντας τις ΠΜΨ και ΕΜ μεταξύ τους, μπορούμε να πούμε ότι η ΠΜΨ είναι πιο αξιόπιστη αλλά πιο αργή από την ΕΜ.

Πίνακας 2.1: Συγκριτική πολυπλοκότητα και αποτελεσματικότητα μεθόδων

	1. ΕΜΑ		2. ΨΜΑ		3. ΠΜΨ	
	απλές	σύνθετες	απλές	σύνθετες	απλές	σύνθετες
offline υπολογισμός	$O(2^N)$	$O(2^{2N})$	O(NW)	$O(m^2NW)$	πολυωνυμικός	
εύρεση επόμενης πρότασης ανά γύρο	O(N)	O(N)	O(N)	O(N)	O(N)	O(N)
πλήθος γύρων	$<2^N$	$<2^N$	<W	<W	<|DL|	<|DL|
αποτέλεσμα	pareto βέλτιστο & ατομικά ορθολογικό		pareto βέλτιστο & ατομικά ορθολογικό		προσεγγιστικό – ίσως μη pareto βέλτιστο & ατομικά ορθολογικό	

	4α. ΕΜ χωρίς ταξινόμηση		4β. ΕΜ με ταξινόμηση	
	απλές	σύνθετες	απλές	σύνθετες
offline υπολογισμός	–	–	O(NlogN)	O(NlogN)
εύρεση επόμενης πρότασης ανά γύρο	O(N)	O(N)	O(1)	O(m)
πλήθος γύρων	<N	<N	<N	<N
αποτέλεσμα	προσεγγιστικό – ίσως μη pareto βέλτιστο & ατομικά ορθολογικό			

2.8. Συνδυασμός απλών και σύνθετων εργασιών.

Είναι δυνατό στον ίδιο τομέα εργασιών να συνυπάρχουν πολλών ειδών εργασίες, απλές και σύνθετες. Υπάρχουν 2 βασικές ιδέες για την αντιμετώπισή τους:

Ι. Οι απλές εργασίες και οι σύνθετες αναλυμένες σε υποεργασίες να θεωρούνται ως δυο ξεχωριστά είδη και να αντιμετωπίζονται με διαφορετικό τρόπο. Για παράδειγμα, η ΕΜ θα μπορούσε να προσαρμοστεί ως εξής: Σε κάθε γύρο να επιλέγεται η συμφερότερη από τις απλές εργασίες και η συμφερότερη από τις υποεργασίες που οδηγεί όμως σε αποδεκτό υποσύνολο. Από τις δυο τους, να επιλέγεται η συμφερότερη. Αν είναι απλή, τότε το υποσύνολο και το κόστος

αντιπάλου υπολογίζονται εύκολα. Αν είναι υποεργασία σύνθετης εργασίας, τότε εφαρμόζεται η αντίστοιχη διαδικασία για σύνθετες.

ΙΙ. Μια άλλη ιδέα είναι να θεωρήσουμε τις απλές εργασίες ως ειδική περίπτωση των σύνθετων και να τις αντιμετωπίζουμε ως τέτοιες. Ειδικότερα θεωρούμε τις απλές εργασίες ως σύνθετες με μοναδιαίο πλήθος υποεργασιών και μόνο πάγιο κόστος. Με τον τρόπο αυτό μπορούν να "αναλυθούν" σε μοναδιαίες κύριες υποεργασίες και να εφαρμοστούν όσα ισχύουν για τις σύνθετες.

Η δεύτερη περίπτωση, δηλαδή η ενιαία αντιμετώπιση απλών και σύνθετων εργασιών την οποία και θα ακολουθήσουμε είναι απλούστερη στην κατανόηση και την υλοποίηση. Ενδέχεται όμως να έχουμε κάποια περιττή επιβάρυνση της διαδικασίας η οποία να αναζητηθεί αμέσως τώρα. Από την προσεκτική μελέτη των αλγορίθμων offline υπολογισμού για τις τέσσερις μεθόδους που συζητήθηκαν, παρατηρούμε ότι πράγματι η αντιμετώπιση των σύνθετων εργασιών είναι κάπως πολυπλοκότερη από αυτή των απλών. Αναλυτικά έχουμε:

Η ΕΜΑ, είναι εκθετική τόσο για απλές όσο και για σύνθετες εργασίες, όμως στην πρώτη περίπτωση έχει πολυπλοκότητα $O(2^N)$, ενώ στη δεύτερη $O(2^{2N})$.

Η ΨΜΑ, είναι ψευδοπολυωνυμική τόσο για απλές όσο και για σύνθετες εργασίες. Στην πρώτη περίπτωση έχει πολυπλοκότητα O(NW), ενώ στη δεύτερη $O(m^2NW)$. Όμως, στις απλές εργασίες, m=1.

Η ΠΜΨ έχει και στις δυο περιπτώσεις πολυωνυμική πολυπλοκότητα και η ΕΜ προαιρετική offline ταξινόμηση πολυπλοκότητας O(NlogN).

Αναφορικά με τον αλγόριθμο εύρεσης επόμενης πρότασης που εφαρμόζεται σε κάθε γύρο, είναι γενικά ελαφρώς πολυπλοκότερος για σύνθετες εργασίες, εφόσον το υποσύνολο του αντιπάλου είναι μη αποδεκτό και χρειάζεται μετατροπή σε αποδεκτό. Αν όμως προσπαθήσουμε να εφαρμόσουμε τον αλγόριθμο για σύνθετες εργασίες σε απλές, τότε δεν θα προκύψουν καθόλου μη αποδεκτά υποσύνολα.

Συμπερασματικά, εφαρμόζοντας σε απλές εργασίες τους αλγορίθμους για σύνθετες, δεν έχουμε ιδιαίτερη επιβάρυνση, εκτός ίσως μόνο από την ΕΜΑ.

Ανακεφαλαίωση.

Στο κεφάλαιο αυτό, κατηγοριοποιήθηκαν καταρχήν οι εργασίες σε απλές και σύνθετες. Στη συνέχεια, εστιάσαμε σε λεπτομέρειες του ΜΠΣ και στις στρατηγικές των πρακτόρων για διαπραγμάτευση σε ΤΠΕ απλών και σύνθετων εργασιών. Κατόπιν, προτάθηκαν και αναλύθηκαν διάφορες μέθοδοι για τον υπολογισμό της πρότασης που κάνει κάθε πράκτορας στον επόμενο γύρο. Η

διαδικασία αυτή είναι ιδιαίτερα πολύπλοκη και χρονοβόρα και εξετάστηκαν διάφορες παραλλαγές για τη βελτίωσή της.

Στο επόμενο κεφάλαιο θα γίνει συζήτηση για θέματα σχεδίασης και υλοποίησης του συστήματος.

3. Θέματα Σχεδίασης & Υλοποίησης.

3.1. Αρχιτεκτονική συστήματος.

Η λειτουργία ενός πράκτορα και η διαδικασία της διαπραγμάτευσης, μπορεί να χωριστεί στα παρακάτω τμήματα: παραγωγή προτάσεων, αποτίμηση προτάσεων, επικοινωνία με τον αντίπαλο & το σύστημα, διασύνδεση με το χρήστη.

3.2. Θέματα σχεδίασης και υλοποίησης.

Για τη σχεδίαση και την υλοποίηση της προσομοίωσης ενός απλού συστήματος διαπραγμάτευσης μεταξύ δύο πρακτόρων, επιλέγεται αντικειμενοστραφής προσέγγιση, εφόσον αντικείμενα και πράκτορες παρουσιάζουν πολλά κοινά σημεία, παρά τις διαφορές τους κυρίως στο θέμα της αυτονομίας[27]. Ειδικότερα, τα αντικείμενα είναι υπολογιστικές οντότητες που ενθυλακώνουν (encapsulate) τα χαρακτηριστικά τους, εκτελούν τις μεθόδους τους και ανταλλάσσουν μηνύματα, όπως και οι πράκτορες ελέγχουν την εσωτερική τους κατάσταση, επιτελούν ενέργειες και επικοινωνούν. Συνεπώς τα αντικείμενα μπορούν να χρησιμοποιηθούν ως βάση για την ανάλυση και υλοποίηση πρακτόρων, με κάποιες πρόσθετες παρεμβολές για την επίτευξη αυτονομίας και ευελιξίας στη συμπεριφορά.

Στο Παράρτημα περιλαμβάνονται τα παρακάτω διαγράμματα δραστηριοτήτων για το σύστημα σε σημειολογία UML

- Διάγραμμα Π.1: Δραστηριότητα «Διαπραγμάτευση».
- Διάγραμμα Π.2: Δραστηριότητα «Αποστολή πρότασης».
- Διάγραμμα Π.3: Δραστηριότητα «Παραλαβή πρότασης».
- Διάγραμμα Π.4: Δραστηριότητα «Παραγωγή πρότασης».

[27] [Wooldridge 2002, κεφ.2]

- Διάγραμμα Π.5: Δραστηριότητα «Αποτίμηση πρότασης».

Επίσης, στο Παράρτημα περιλαμβάνεται το διαγράμμα κλάσεων (Διάγραμμα Π.7), καθώς και η περιγραφή των χρησιμοποιούμενων κλάσεων.

Για την υλοποίηση επιλέχτηκε η Java, που είναι αντικειμενοστραφής γλώσσα προγραμματισμού, ανεξάρτητη πλατφόρμας, ευρέως χρησιμοποιούμενη σε πολλές εφαρμογές γενικού σκοπού και σε εφαρμογές πρακτόρων. Η Java παρέχει τις δυνατότητες δημιουργίας νημάτων, συγχρονισμού, γραφικής διασύνδεσης με το χρήστη και εύκολης διαχείρισης συμβολοσειρών. Για τους παραπάνω λόγους η Java κρίνεται επαρκής για τις ανάγκες του προς ανάπτυξη πρωτοτύπου.

Ζητούμενο είναι να εκτελούνται παράλληλα τόσο οι δύο πράκτορες όσο και το κυρίως σύστημα, που παίζει το ρόλο του κεντρικού κόμβου – συντονιστή. Για το λόγο αυτό το κυρίως πρόγραμμα δημιουργεί δυο νήματα (threads), ένα για κάθε πράκτορα, κάνοντας χρήση την κλάση Thread της Java.

Για την επικοινωνία μεταξύ πρακτόρων, χρησιμοποιείται μια ειδική αποθήκη στην οποία τοποθετούνται και από την οποία διαβάζονται δεδομένα, δηλαδή τα μηνύματα που ανταλλάσσουν οι πράκτορες. Για να μην μπερδεύονται τα μηνύματα, κάθε πράκτορας χρησιμοποιεί μία αποθήκη για να γράφει από την οποία διαβάζει ο άλλος πράκτορας και μια δεύτερη αποθήκη για να διαβάζει ό,τι έγραψε ο αντίπαλος. Για να μην πραγματοποιείται η αποστολή και η λήψη με διαφορετική ταχύτητα και δημιουργούνται προβλήματα, λαμβάνει χώρα συγχρονισμός, με τη χρήση wait() και notify().

Τα μηνύματα που ανταλλάσσονται κατά τη διαπραγμάτευση περιλαμβάνουν προτάσεις. Οι προτάσεις, αν και η αναπαράσταση και διαχείρισή τους σε κάθε πράκτορα γίνεται με τη μορφή αντικειμένων, ωστόσο δεν ανταλλάσσονται ως αντικείμενα αλλά ως συμβολοσειρές. Κι αυτό γιατί, αν και είναι πιο πολύπλοκο και χρονοβόρο, ωστόσο είναι πιο γενικό. Έτσι, ακόμη και αν κάθε πράκτορας είχε οποιαδήποτε διαφορετική εσωτερική αναπαράσταση και μια διεπαφή για τη μετατροπή σε συμβολοσειρά, το πρόγραμμα θα εξακολουθούσε να δουλεύει. Μια συμβολοσειρά, επίσης, μεταφέρεται ευκολότερα μέσω ενός δικτύου, αν υποθέσουμε ότι οι πράκτορες ήταν γεωγραφικά απομακρυσμένοι και όχι νήματα του ίδιου προγράμματος και υπολογιστή.

Πολλές φορές κατά την υλοποίηση χρειάζεται να αναφερθούμε σε ένα υποσύνολο εργασιών. Κάθε εργασία χαρακτηρίζεται από έναν αριθμό, π.χ. οι εργασίες 0,2,5,8,10,15. Το υποσύνολο μπορεί να αναπαρασταθεί με τους παρακάτω τρόπους:

- Ως String "0,2,5,8,10,15", που είναι όμως δύσκολη η διαχείρισή του.
- Ως πίνακας ακεραίων {0,2,5,8,10,15}, που πιάνει πολύ χώρο.
- Ως boolean πίνακας {T,F,T,F,F,T,F,F,T,F,T,F,F,F,F,T}, όπου T δηλώνει την ύπαρξη του αριθμού και F την απουσία του. Επίσης, πιάνει πολύ χώρο.
- Μπορούμε να εκμεταλλευτούμε την τελευταία ιδέα και να αντικαταστήσουμε το T με 1 και το F με 0. Έτσι προκύπτει ένας δυαδικός αριθμός που εύκολα μετατρέπεται στον αντίστοιχο δεκαδικό ακέραιο. Πλέον το υποσύνολο εργασιών αναπαρίσταται μοναδικά με έναν ακέραιο από 0 … 2^N, όπου N το πλήθος των εργασιών. Όταν θέλουμε να δούμε αν μια εργασία i ανήκει στο υποσύνολο s, χρησιμοποιούμε το δυαδικό τελεστή & της Java στη σχέση $s\&2^i != 0$.

Στην υλοποίηση χρησιμοποιούνται offline τεχνικές έτσι ώστε κάποια πράγματα να υπολογίζονται όχι κάθε φορά που χρειάζονται, ξεκινώντας από το μηδέν, και ενώ οι πράκτορες τα περιμένουν, αλλά μία φορά στην αρχή. Αυτό εφαρμόζεται στις μεθόδους ΕΜΑ, ΨΜΑ και ΠΜΨ για τον υπολογισμό υποσυνόλων και κόστους, όπως θα δούμε στη συνέχεια. Επίσης, επειδή οι διάφορες δυνάμεις του 2 χρησιμοποιούνται αρκετές φορές κατά τη διάρκεια του προγράμματος, μπορούν να υπολογιστούν μια φορά στην αρχή, να αποθηκευτούν σε πίνακα και να ανακτώνται ανά πάσα στιγμή με τη μέθοδο get2e(), έτσι ώστε $get2e(i) = 2^i$.

Για την επικοινωνία με το χρήστη χρησιμοποιούνται κλάσεις του Swing. Πιο συγκεκριμένα, ανοίγει ένα παράθυρο στην αρχή του προγράμματος για εισαγωγή στοιχείων. Στη συνέχεια, εμφανίζεται ένα παράθυρο για ενημέρωση του χρήστη για την πορεία της διαπραγμάτευσης, που περιλαμβάνει τις προτάσεις που στέλνονται και παραλαμβάνονται, καθώς και η αποτίμηση και τη γραφική απεικόνισή τους.

Τέλος, υπάρχει προαιρετικά στο πρόγραμμα η δυνατότητα για την εισαγωγή μιας τεχνητής καθυστέρησης σε sec, για να μπορεί ο χρήστης να παρακολουθεί πιο σιγά την πορεία της διαπραγμάτευσης.

3.3. Αναπαράσταση πληροφορίας.

Καταρχήν θα εξηγηθούν μερικές συμβάσεις που χρησιμοποιούνται παρακάτω.

Θεωρούμε ότι έχουμε δύο πράκτορες 0 και 1, οι οποίοι λένε την αλήθεια και έχουν παντογνωσία. Οι προτάσεις των πρακτόρων έχουν τη μορφή "είδος_πρότασης αποστολέας παραλήπτης deal", π.χ. offer 0 1 <{t0},{t1}>.

Οι πράκτορες χρησιμοποιούν το ίδιο πρωτόκολλο, το Μονοτονικό Πρωτόκολλο Συμβιβασμού (ΜΠΣ).

Επίσης, χρησιμοποιούν τις ίδιες ή διαφορετικές μεθόδους και πολιτικές.

Χρησιμοποιούμενες μέθοδοι για την εύρεση επόμενης καλύτερης πρότασης είναι: 1 – εκθετική μέθοδος ακρίβειας (ΕΜΑ), 2 – ψευδοπολυωνυμική μέθοδος ακρίβειας (ΨΜΑ), 3 – προσεγγιστική μέθοδος ψαλιδίσματος (ΠΜΨ) και 4 – ευρετική μέθοδος (ΕΜ).

Χρησιμοποιούμενες πολιτικές υποχώρησης είναι: 1 – στρατηγική Zeuthen, 2 –συνεχής υποχώρηση, 3 – μηδενική υποχώρηση, 4 – τυχαία υποχώρηση.

Επίσης, κάθε πράκτορας κάνει σε κάθε γύρο την ελάχιστη δυνατή υποχώρηση. Πιο συγκεκριμένα, αν χρησιμοποιεί την πολιτική 2, επιλέγει την αμέσως επόμενη καλύτερη πρόταση. Αν χρησιμοποιεί τη στρατηγική Zeuthen, ελέγχει για κάθε μια από τις αμέσως επόμενες καλύτερες προτάσεις, ποια αλλάζει την ισορροπία του ρίσκου και την προτείνει.

Αν στο τελευταίο βήμα συμφωνήσουν και οι δυο με την πρόταση του αντιπάλου, τότε επιλέγεται η πρόταση που μεγιστοποιεί το γινόμενο των χρησιμοτήτων των δυο πρακτόρων. Σε περίπτωση ισότητας, επιλέγεται αυτή που μεγιστοποιεί το άθροισμα των χρησιμοτήτων. Σε περίπτωση ισότητας, επιλέγεται αυθαίρετα μια από τις δυο.

Η αρχική αλληλεπίδραση $<T_0,T_1>$ μπορεί είτε να καθορίζεται από το χρήστη είτε να προκύπτει από το πρόγραμμα με τυχαίο τρόπο.

Αναφορικά με τα είδη των χρησιμοποιούμενων εργασιών, διακρίνουμε τρεις περιπτώσεις:

Α. Θεωρούμε ότι έχουμε N απλές διακριτές εργασίες $(t_0,t_1,\ldots t_{N-1})$. Σε κάθε εργασία αντιστοιχεί ένα θετικό, μη μηδενικό κόστος c_i.

Β. Θεωρούμε ότι έχουμε να διαχειριστούμε μια σειρά από Q σύνθετες εργασίες τις $\{f_0(m_0), f_1(m_1), \ldots, f_{N-1}(m_{Q-1})\}$. Υπενθυμίζεται η σύμβαση που έχουμε ήδη κάνει ότι μια εργασία f(πλήθος, ποσότητα) μετατρέπεται από το χρήστη στις f_είδος1(ποσότητα), f_είδος2(ποσότητα), …, έτσι ώστε να υπάρχει μία μόνο μεταβλητή, η ποσότητα. Η εργασία f_0 έχει m_0 υποεργασίες, η f_1 έχει m_1

υποεργασίες, κοκ. Ισοδύναμα, έχουμε τις υποεργασίες $\{f_0(1), f_0(1), \ldots, f_0(1), f_1(1), \ldots, f_1(1), \ldots, f_{N-1}(1), \ldots, f_{Q-1}(1)\}$ ή απλούστερα $\{f_0, f_0, \ldots, f_0, f_1, \ldots, f_1, \ldots, f_{Q-1}, \ldots, f_{Q-1}\}$. Η πρώτη υποεργασία κάθε εργασίας είναι κύρια, με κόστος c0+c1, και όλες οι υπόλοιπες, κόστους c1, είναι δευτερεύουσες. Όλες οι δευτερεύουσες υποεργασίες της ίδιας εργασίας θεωρούνται ισοδύναμες. Έστω N το πλήθος όλων των υποεργασιών, δηλαδή $m_0 + m_1 + \ldots + m_{Q-1}$. Οι υποεργασίες πλέον έχουν ενιαία αρίθμηση από 0 έως N-1 για τις εσωτερικές ανάγκες του προγράμματος. Η αντιμετώπιση των σύνθετων εργασιών γίνεται με μετατροπή τους σε υποεργασίες, αντιμετώπιση των υποεργασιών με τρόπο ανάλογο με τις απλές εργασίες, με κάποιες ειδικές συνθήκες όμως, και τέλος ανασύνθεση των σύνθετων εργασιών.

Παράδειγμα. Για την εργασία $\{f_0(4)\}$ η αρίθμηση είναι 0-f0, 1-f0, 2-f0, 3-f0. Το υποσύνολο $\{f_0(3)\}$ είναι {f0,f0,f0} που μπορεί να αντιστοιχεί σε {0,1,2} ή {0,1,3} ή {0,2,3}. Τα τρία είναι ισοδύναμα. Το {1,2,3} είναι λάθος γιατί περιέχει μόνο δευτερεύουσες εργασίες και πρέπει να αντικατασταθεί από ένα από τα προηγούμενα. □

Γ. Θεωρούμε ότι έχουμε συνδυασμό απλών και σύνθετων εργασιών. Στην περίπτωση αυτή τις αντιμετωπίζουμε όλες ως σύνθετες, όπως περιγράψαμε στο προηγούμενο κεφάλαιο.

Στη συνέχεια θα δούμε πώς αναπαριστούμε και διαχειριζόμαστε την πληροφορία και ειδικότερα τους πράκτορες, τις εργασίες, τις προτάσεις και τις αλληλεπιδράσεις.

Αναπαράσταση πρακτόρων και εργασιών.

Τα στοιχεία ταυτότητας πράκτορα περιλαμβάνονται στην κλάση AgentID. Κάθε πράκτορας έχει όνομα που περιέχεται στο *agentName* String[2]. Εσωτερικά αναπαρίσταται με ένα κωδικό 0 ή 1. Ο κωδικός πράκτορα αποθηκεύεται στη μεταβλητή *myself* και ο κωδικός αντιπάλου στη μεταβλητή *opponent*.

Σενάριο 3.1α.[28]

> Έστω 2 πράκτορες {a,b}. Η αντιστοίχιση είναι a↔0, b↔1 και το διάνυσμα είναι agentName = {a, b}. Για τον πράκτορα a είναι myself = 0 και opponent = 1. Για τον πράκτορα b είναι myself = 1 και opponent = 0. □

[28] Το σενάριο 3.1 αναφέρεται σε απλές εργασίες και παρουσιάζεται σταδιακά. Περιλαμβάνει τα υποσενάρια 3.1α, 3.1β, κοκ.

Τα στοιχεία των εργασιών περιλαμβάνονται στην κλάση Tasks. Κάθε απλή εργασία έχει όνομα που περιέχεται στο *name* String[N] και κόστος που περιέχεται στο *cost* int[N]. Αναπαρίσταται εσωτερικά με ένα κωδικό, έναν αύξοντα αριθμό.

Σενάριο 3.1β.

> Έστω 4 εργασίες $\{t_0, t_1, t_2, t_3\}$ με κόστος {1, 3, 5, 10}. Αντιστοιχίζεται $t_0 \leftrightarrow 0$, $t_1 \leftrightarrow 1$, $t_2 \leftrightarrow 2$, $t_3 \leftrightarrow 3$ και τα διανύσματα είναι name = {t0, t1, t2, t3} και cost = {1, 3, 5, 10}. □

Σημειώνεται ότι για την επικοινωνία με το χρήστη χρησιμοποιούνται τα πραγματικά ονόματα των πρακτόρων και εργασιών, ενώ εσωτερικά στο πρόγραμμα και για την επικοινωνία των πρακτόρων μεταξύ τους χρησιμοποιούνται οι κωδικοί. Για παράδειγμα, για τις δυο περιπτώσεις έχουμε "offer a b <{t0 }{t1 t2 t3 }>" και "offer 0 1 {0 }{1 2 3 }" αντίστοιχα.

Κάθε σύνθετη εργασία έχει όνομα που περιέχεται στο *name* String[Q]. Το διάνυσμα *number* είναι int[Q] και περιέχει την ποσότητα κάθε εργασίας, δηλαδή το πλήθος των υποεργασιών της.

Σενάριο 3.2α.[29]

> Έστω οι εργασίες $\{f_0(2), f_1(1), f_2(1)\}$. Το διάνυσμα name είναι {f0, f1, f2}, ενώ number = {2, 1, 1}. □

Ο πίνακας *costI* είναι int[2][Q] και περιέχει το κόστος κάθε εργασίας. Στην πρώτη γραμμή αποθηκεύονται τα c_{i0} και στη δεύτερη τα c_{i1}.

Σενάριο 3.2β.

> Έστω οι $\{f_0(2), f_1(1), f_2(1)\}$ με κόστος $c_0(n)=1+n$, $c_1(n)=1+n$, $c_2(n)=1+n$. Ο πίνακας costI είναι {{1,1,1},{1,1,1}}. □

Ανάλυση σύνθετων εργασιών σε υποεργασίες.

Οι παραπάνω πληροφορίες χρησιμοποιούνται για τη μετατροπή των σύνθετων εργασιών σε ισοδύναμες υποεργασίες. Τίθεται το ερώτημα ποιος κάνει αυτή τη μετατροπή. Επειδή διαδικασία είναι η ίδια και για τους δυο πράκτορες, είναι καλύτερα να την κάνει το σύστημα, δηλαδή η κλάση MAS, και όχι καθένας ξεχωριστά. Η μετατροπή περιλαμβάνει υπολογισμό του N, δηλαδή του πλήθους όλων των υποεργασιών και την κατασκευή των διανυσμάτων κόστους υποεργασιών και εξαρτήσεων.

Το διάνυσμα *cost* είναι int[N] και περιέχει το κόστος κάθε υποεργασίας.

---

[29] Το σενάριο 3.2 αναφέρεται σε σύνθετες εργασίες και παρουσιάζεται σταδιακά. Περιλαμβάνει τα υποσενάρια 3.2α, 3.2β, κοκ.

Σενάριο 3.2γ.

Έστω οι $\{f_0(2), f_1(1), f_2(1)\}$ με κόστος $c_0(n)=1+n$, $c_1(n)=1+n$, $c_2(n)=1+n$. Ισοδύναμα έχουμε τις υποεργασίες $\{f_0, f_0, f_1, f_2\}$. Είναι cost = {2, 1, 2, 2}. □

Το διάνυσμα *dependency* είναι int[N] αναπαριστά τις εξαρτήσεις. Περιέχει τον κωδικό της υποεργασίας από την οποία είναι δευτερεύουσα η τρέχουσα υποεργασία. Αν αυτή είναι ανεξάρτητη, το κελί γίνεται -1. Παρατηρούμε ότι για εργασίες της μορφής που περιγράψαμε, μια υποεργασία εξαρτάται το πολύ από μία άλλη, την κύριά της.

Σενάριο 3.2δ.

Έστω οι $\{f_0(2), f_1(1), f_2(1)\}$ με κόστος $c_0(n)=1+n$, $c_1(n)=1+n$, $c_2(n)=1+n$. Ισοδύναμα έχουμε τις υποεργασίες $\{f_0, f_0, f_1, f_2\}$. Διάνυσμα εξαρτήσεων dependency = {-1, 0, -1, -1}. □

Σημειώνεται ότι για την επικοινωνία με το χρήστη χρησιμοποιούνται τα ονόματα των σύνθετων εργασιών και σε παρένθεση το πλήθος των υποεργασιών αν είναι μεγαλύτερο του ένα. Εσωτερικά στο πρόγραμμα και για την επικοινωνία των πρακτόρων μεταξύ τους χρησιμοποιούνται οι κωδικοί των υποεργασιών. Για παράδειγμα, για τις δυο περιπτώσεις έχουμε "offer a b <{f0(2) }{f1 f2 }>" και "offer 0 1 {0 1 }{2 3 }" αντίστοιχα.

Ακολουθούν η διαδικασία ανάλυσης σύνθετων εργασιών σε υποεργασίες, καθώς και η περιγραφή των συναρτήσεων αλληλεξαρτήσεων.

Αλγόριθμος 3.1: Ανάλυση σύνθετων εργασιών σε υποεργασίες

```
for (i=0,N=0; i<Q; i++)
    N += number[i];
cost = new int[N];
dependency = new int[N];
for (i=0,j=0,nj=0; i<N; i++) {
    if (nj==0) { // master subtask
        cost[i] = costI[0][j] + costI[1][j];
        dependency[i] = -1;
        master = i;
    }
    else { // slave subtask
        cost[i] = costI[1][j];
        dependency[i] = master;
    }
    nj++;
    if (nj == number[j]) {
        j++;
        nj = 0;
    }
}
```

Αλγόριθμος 3.2: Συναρτήσεις αλληλεξαρτήσεων υποεργασιών

```
boolean isMaster (int i) { // true, if subtask i is master
    return dependency[i] == -1;
}
boolean isSlave (int j) { // true, if subtask j is slave
    return dependency[j] != -1;
}
boolean isSlave (int j, int i) { // true, if subtask j is slave of i
    return dependency[j] == i;
}
int getMaster (int j) { // returns master of subtask j
    return dependency[j];
}
boolean hasMasterInDeal (int j, boolean dpart[]) {
// true, if master of subtask j exists in deal
    return dpart[getMaster(j)];
}
int getSlaveFromDeal (int i, boolean dpart[]) {
// returns the first slave of subtask i it finds in deal
    for (j=i; j<Q; j++)
        if (dependency[j] == i && dpart[j])
            return j;
    return -1;
}
```

Αναπαράσταση προτάσεων.

Στη συνέχεια, θα δούμε πως γίνεται η αναπαράσταση προτάσεων (deals).

Κατά τη διαπραγμάτευση, κατά σύμβαση πρώτο μπαίνει πάντα το υποσύνολο των εργασιών του πράκτορα 0 και μετά του 1, δηλαδή d = $<D_0, D_1>$.

Στις απλές εργασίες, μια πρόταση αναπαρίσταται με ένα πίνακα boolean[2][N]. Η πρώτη γραμμή αντιστοιχεί στο D_0 και η δεύτερη στο D_1. Το κελί [0][i] είναι true, αν η εργασία i ανατίθεται στον πράκτορα 0. Το κελί [1][i] είναι true, αν η εργασία i ανατίθεται στον πράκτορα 1.

Σημειώνεται ότι ειδικά για τις απλές εργασίες η μια γραμμή είναι πάντα συμπληρωματική της άλλης, δηλαδή d[1][i] = !d[0][i]. Αυτό ισχύει γιατί οι προτάσεις που ανήκουν στο σύνολο διαπραγμάτευσης, είναι pareto βέλτιστες κι επομένως τα D_0 και D_1 δεν έχουν κοινά στοιχεία. Μια εργασία δηλαδή θα την αναλάβει είτε ο ένας είτε ο άλλος. Μάλιστα, για τις απλές εργασίες, για την αναπαράσταση της πρότασης που κάνει ένας πράκτορας, θα αρκούσε ένα μονοδιάστατο διάνυσμα, του οποίου τα κελιά με τιμή true να αντιστοιχούν κατά σύμβαση στις εργασίες του ενός και τα κελιά με false στις εργασίες του άλλου.

Χρησιμοποιείται όμως δισδιάστατος πίνακας για να υπάρχει ομοιομορφία με τις σύνθετες εργασίες, για τις οποίες η παραπάνω σύμβαση δεν ισχύει.

Σενάριο 3.1γ.

Έστω οι εργασίες $\{t_0, t_1, t_2, t_3\}$ με κόστος {1, 3, 5, 10}. Η πρόταση <{t0,t1,t3}{t2}> αντιστοιχεί σε d = {{T, T, F, T}, {F, F, T, F}}. □

Στις σύνθετες εργασίες, γίνεται πρώτα η μετατροπή τους σε N υποεργασίες. Η πρόταση αναφέρεται στις N υποεργασίες και αναπαρίσταται επίσης με ένα πίνακα boolean[2][N]. Η πρώτη γραμμή αντιστοιχεί στο D_0 και η δεύτερη στο D_1. Το κελί [0][i] είναι true, αν η υποεργασία i ανατίθεται στον πράκτορα 0. Το κελί [1][i] είναι true, αν η υποεργασία i ανατίθεται στον πράκτορα 1.

Σενάριο 3.2ε.

Έστω οι $\{f_0(2), f_1(1), f_2(1)\}$ με κόστος $c_0(n)=1+n$, $c_1(n)=1+n$, $c_2(n)=1+n$. Ισοδύναμα έχουμε τις υποεργασίες $\{f_0, f_0, f_1, f_2\}$. Έστω η πρόταση d1 $<\{f_0(2)\}$, $\{f_1(1), f_2(1)\}>$. Ισχύει d1 = {{T, T, F, F}, {F, F, T, T}}. □

Για τις σύνθετες εργασίες, η μια γραμμή του πίνακα που αναπαριστά την πρόταση δεν είναι πάντα συμπληρωματική της άλλης. Έστω $<D_M, D_O>$. Αν είναι γνωστό το D_M ή d[myself], τότε μπορεί να υπολογιστεί το D_O ή d[opponent] ως συμπλήρωμα του D_M ή d[myself], αρκεί να προκύπτει αποδεκτό υποσύνολο. Διαφορετικά πρέπει να μετατραπεί στο αντίστοιχο αποδεκτό. Πιο συγκεκριμένα, αν t_i είναι μια δευτερεύουσα υποεργασία του D_O και η αντίστοιχη κύρια δεν ανήκει στο D_O, τότε από το D_O πρέπει να αφαιρεθεί η δευτερεύουσα υποεργασία και να προστεθεί η κύρια.

Αλγόριθμος 3.3: Υπολογισμός D_O (d[opponent]) από το D_M (d[myself])

```
boolean[][] getComplementary() {
    for (i=0; i<N; i++)
        d[opponent][i] = !d[myself][i];
    for (i=0; i<N; i++)
        if (isSlave(i) && d[opponent][i] && not d[opponent][getMaster(i)]) {
            d[opponent][getMaster(i)] = true;
            d[opponent][i] = false;
        }
    return d;
}
```

Σενάριο 3.2στ.

Έστω οι $\{f_0(2), f_1(1), f_2(1)\}$ με κόστος $c_0(n)=1+n$, $c_1(n)=1+n$, $c_2(n)=1+n$. Ισοδύναμα έχουμε τις υποεργασίες $\{f_0, f_0, f_1, f_2\}$. Έστω d2 $<\{f_0(1), f_1(1)\}$, $\{f_0(1), f_2(1)\}>$. Κανονικά θα έπρεπε d2 = {{T, F, T, F}, {F, T, F, T}}. Όμως, η δευτερεύουσα υποεργασία 1 περιλαμβάνεται στο D_O σε αντίθεση με την κύριά

της 0. Η 1 δεν μπορεί να σταθεί μόνη της χωρίς την 0, οπότε η υποεργασία 0 την αντικαθιστά. Τελικά ισχύει d2 = {{T, F, T, F}, {T, F, F, T}}. □

Αλγόριθμος 3.4: Υπολογισμός κόστους μιας πρότασης

```
int getCost (boolean[] dRow) { // returns cost of a deal for an agent
    for (i=0,c=0; i<N; i++)
        if (dRow[i]) c += cost[i];
    return c;
}
```

Σημειώνεται ότι για μια πρόταση d[][], η μέθοδος καλείται ως εξής: costM = getCost(d[myself]) και costO = getCost(d[opponent]).

Σενάριο 3.1δ.

Έστω οι εργασίες $\{t_0, t_1, t_2, t_3\}$ με κόστος {1, 3, 5, 10}. Για d = {{F, F, T, F}, {T, T, F, T}}, το κόστος του πράκτορα 0 είναι cost(d,0) = cost[2] = 5. Το κόστος του 1 είναι cost(d,1) = cost[0] + cost[1] + cost[3] = 1+3+10 =14. □

Σενάριο 3.2ζ.

Έστω οι $\{f_0(2), f_1(1), f_2(1)\}$ με κόστος $c_0(n)=1+n$, $c_1(n)=1+n$, $c_2(n)=1+n$. Ισοδύναμα έχουμε τις υποεργασίες $\{f_0, f_0, f_1, f_2\}$ και costI = {2, 1, 2, 2}.
Αν d1 = $\{\{f_0(2)\}, \{f_1(1), f_2(1)\}\}$, τότε cost(d1,0) = cost[0] + cost[1] = 3, cost(d1,1) = cost[2] + cost[3] = 4.
Αν d2 = $\{\{f_0(1), f_1(1)\}, \{f_0(1), f_2(1)\}\}$, τότε cost(d2,0) = cost[0] + cost[2] = 4, cost(d2,1) = cost[0] + cost[3] = 4. □

Μια πρόταση αναπαρίσταται εσωτερικά σε κάθε πράκτορα ως αντικείμενο Proposal. Κατά την ανταλλαγή προτάσεων μεταξύ πρακτόρων και την ενημέρωση του χρήστη για την πορεία της διαπραγμάτευσης, μετατρέπεται σε String της μορφής "type sender receiver deal".

Σενάριο 3.1ε.

Έστω οι εργασίες $\{t_0, t_1, t_2, t_3\}$ με κόστος {1, 3, 5, 10}. Έστω η πρόταση $<\{t_0, t_1, t_3\}\ \{t_2\}>$, δηλαδή d = {{T, T, F, T}, {F, F, T, F}}. Για ενημέρωση του χρήστη μετατρέπεται στη συμβολοσειρά "<{t0 t1 t3 }{t2 }>". Για μεταφορά στον αντίπαλο μετατρέπεται στη συμβολοσειρά "{0 1 3 }{2 }", από το οποίο ο αντίπαλος αναπαράγει το d = {{T, T, F, T}, {F, F, T, F}}. □

Σενάριο 3.2η.

Έστω η πρόταση $<\{f_0(2)\}\ \{f_1(1), f_2(1)\}>$ ή ισοδύναμα με χρήση υποεργασιών $<\{f_0\ f_0\}\ \{f_1, f_2\}>$. Ισχύει d = {{T, T, F, F}, {F, F, T, T}}. Για ενημέρωση του χρήστη μετατρέπεται σε <{f0(2) } {f1(1) f2(1) }>. Για μεταφορά στον αντίπαλο μετατρέπεται σε {0 1 }{2 3 }, από το οποίο ο αντίπαλος αναπαράγει το d = {{T, T, F, F}, {F, F, T, T}}. □

Αλγόριθμος 3.5: Μετατροπή πρότασης σε συμβολοσειρά κατά την επικοινωνία πρακτόρων

```
s0 = "",  s1 = "";
for (i=0; i<d[0].length; i++){
    if (d[0][i])
        s0 += i+" ";
    if (d[1][i])
        s1 += i+" ";
}
return "{"+s0+"}{"+s1+"}";
```

Αλγόριθμος 3.7: Μετατροπή πρότασης σε συμβολοσειρά για ενημέρωση χρήστη

```
s0 = "";
s1 = "";
n0 = 0;
n1 = 0;
if (complex)
    for (i=0; i<d[0].length; i++) {
        if (d[0][i])
            s0 += getName(i) + " ";
        if (d[1][i])
            s1 += getName(i) + " ";
    }
else
    for (i=0,j=0,nj=0; i<d[0].length; i++) {
        if (d[0][i])
            n0++;
        if (d[1][i])
            n1++;
        nj++;
        if (nj == getNumber(j)) {
            if (n0 == 1)
              s0 += getName(j) + " ";
            else if (n0 > 0)
              s0 += getName(j) + "(" + n0 + ") ";
            if (n1 == 1)
              s1 += getName(j) + " ";
            else if (n1 > 0)
              s1 += getName(j) + "(" + n1 + ") ";
            j++;
            nj = 0;
            n0 = 0;
            n1 = 0;
        }
    }
return "<{"+s0+"}{"+s1+"}>";
```

Αλγόριθμος 3.8: Μετατροπή συμβολοσειράς σε πρόταση κατά την επικοινωνία πρακτόρων

```
for (b=end+1, N=0; b<s.length(); b++)
    if (s.charAt(b) == ' ')
        N++;
// the number of tasks/subtasks is the same with the number of spaces ' '
d = new boolean[2][N];
if (!type.equals("withdraw")) {
    for (ag=0; ag<=1; ag++) {
        for (i=0; i<N; i++)
            d[ag][i] = false;
        begin = s.indexOf("{",end)+1;
        end = s.indexOf("}",begin);
        b = s.indexOf(" ",begin);
        while (b<end && b>0) {
            i = Integer.parseInt(s.substring(begin,b));
            d[ag][i] = true;
            begin = b+1;
            b = s.indexOf(" ",begin);
        }
    }
}
```

Αναπαράσταση αλληλεπιδράσεων.

Στη συνέχεια, θα δούμε πως γίνεται η αναπαράσταση αλληλεπιδράσεων (encounters).

Αναφορικά με τις απλές εργασίες, για την αναπαράσταση αλληλεπιδράσεων, χρησιμοποιούμε για κάθε πράκτορα ένα δυσδιάστατο πίνακα ie int[2][N]. Στην πρώτη γραμμή περιέχεται το πλήθος (0 ή 1) των υποεργασιών για κάθε εργασία του πράκτορα 0 και στη δεύτερη αντίστοιχα για τον πράκτορα 1.

Κάνουμε την παρατήρηση ότι σε αντίθεση με τις προτάσεις εδώ οι δυο γραμμές είναι απαραίτητες, γιατί οι αλληλεπιδράσεις δεν είναι pareto βέλτιστες και η ίδια εργασία μπορεί να έχει ανατεθεί και στους δυο πράκτορες ταυτόχρονα. Στις απλές εργασίες ο πίνακας ie παίρνει μόνο τιμές 0 και 1 και θα μπορούσε να οριστεί ως boolean[2][N] όπως και στην αναπαράσταση προτάσεων (deals). Επιλέγουμε όμως τον int[2][N] για να υπάρχει ομοιομορφία με τις σύνθετες εργασίες.

Σενάριο 3.1στ.

> Έστω οι εργασίες $\{t_0, t_1, t_2, t_3\}$ με κόστος {1, 3, 5, 10}. Η αρχική αλληλεπίδραση $<\{ t_0, t_1, t_2, t_3\}\ \{t_2, t_3\}>$ αντιστοιχεί στο ie = {{1, 1, 1, 1},{0, 0, 1, 1}}. □

Για τις σύνθετες εργασίες, η αναπαράσταση αλληλεπίδρασης γίνεται χωρίς τη μετατροπή της σε υποεργασίες. Η πληροφορία αυτή δεν χρειάζεται να είναι αποθηκευμένη κάπου, γιατί η αρχική αλληλεπίδραση χρησιμοποιείται μόνο στην αρχή κατά τον υπολογισμό κόστους. Έτσι, χρησιμοποιείται ένας πίνακα ie int[2][Q], διάστασης Q. Στην πρώτη γραμμή αποθηκεύεται το T_0 και στη δεύτερη το T_1. Ειδικότερα, σε κάθε κελί ie[ag][i] περιέχεται το πλήθος των υποεργασιών της εργασίας i που έχουν ανατεθεί στον πράκτορα ag της αντίστοιχης γραμμής.

Σενάριο 3.2θ.

Έστω οι $\{f_0(2), f_1(1), f_2(1)\}$ με κόστος $c_0(n)=1+n$, $c_1(n)=1+n$, $c_2(n)=1+n$. Ισοδύναμα έχουμε τις υποεργασίες $\{f_0, f_0, f_1, f_2\}$. Έστω η αρχική αλληλεπίδραση $<\{f_0(2), f_1(1), f_2(1)\},\{f_0(2), f_1(1), f_2(1)\}>$. Τότε ie = {{2, 1, 1}, {2, 1, 1}}. □

Αλγόριθμος 3.8: Μετατροπή αλληλεπίδρασης σε συμβολοσειρά

```
s0 = "", s1 = "";
for (i=0; i<ie[0].length; i++) {
    if (ie[0][i] == 1)
        s0 += getName(i) + " ";
    else if (ie[0][i] > 1)
        s0 += getName(i) + "(" + ie[0][i] + ")" + " ";
    if (ie[1][i] == 1)
        s1 += getName(i) + " ";
    else if (ie[1][i] > 1)
        s1 += getName(i) + "(" + ie[1][i] + ")" + " ";
}
return "<{"+s0+"}{"+s1+"}>";
```

Αλγόριθμος 3.9: Υπολογισμός κόστους αλληλεπίδρασης

```
int getCost (int[] erow) { // returns cost of a encounter for an agent
    c = 0;
    if (!complex) {
        for (i=0; i<N; i++)
            if (erow[i] != 0)
                    c += cost[i];
    }
    else {
        for (i=0; i<Q; i++)
            if (erow[i] != 0)
                    c += costI[0][i]+costI[1][i]* erow[i];
    }
    return c;
}
```

Με χρήση του παραπάνω έχουμε ότι το κόστος αρχικής αλληλεπίδρασης για τον πράκτορα είναι icM = getCost(ie[myself]) και για τον αντίπαλο icO = getCost (ie[opponent]).

Σενάριο 3.1ζ.

Έστω οι εργασίες $\{t_0, t_1, t_2, t_3\}$ με κόστος {1, 3, 5, 10}. Αν αρχική αλληλεπίδραση $<\{t_0, t_1, t_2, t_3\},\{t_2, t_3\}>$, ισχύει για τον πράκτορα 0: icM = cost[0] + cost[1] + cost[2] + cost[3] = 19 και icO = cost[2] + cost[3] = 15. □

Σενάριο 3.2ι.

Έστω οι $\{f_0(2), f_1(1), f_2(1)\}$ με κόστος $c_0(n)=1+n$, $c_1(n)=1+n$, $c_2(n)=1+n$. Ισοδύναμα έχουμε τις υποεργασίες $\{f_0, f_0, f_1, f_2\}$. Έστω η αρχική αλληλεπίδραση $<\{f_0(2), f_1(1), f_2(1)\}, \{f_0(2), f_1(1), f_2(1)\}>$, ισοδύναμα <{f0, f0, f1, f2},{f0, f0, f1, f2}>. Τότε cost(ieM) = 7, cost(ieO) = 7. □

Αλγόριθμος 3.10: Υπολογισμός χρησιμότητας πρότασης

```
int getUtility (boolean[][] d, int ag) {
    return getCost(ie[ag]) – getCost(d[ag]);
}
```

Σενάριο 3.1η.

Έστω οι εργασίες $\{t_0, t_1, t_2, t_3\}$ με κόστος {1, 3, 5, 10}. Αν ο πράκτορας 0 προτείνει d0 = <{t0, t1, t3}, {t2}>, τότε έχουμε utility(d0, 0) = cost(e, 0) – cost(d, 0) = 19 – 14 = 5. Επίσης, utility(d0, 1) = cost(e, 1) – cost(d, 1) = 15 – 5= 10. □

Ελάχιστη δυνατή υποχώρηση.

Στο σημείο αυτό θα κάνουμε ένα σχόλιο για τη στρατηγική Zeuthen και την ελάχιστη δυνατή υποχώρηση. Κανονικά, όταν ο πράκτορας χρησιμοποιεί τη στρατηγική Zeuthen, ελέγχει το ρίσκο το δικό του και του αντιπάλου και στο νέο γύρο k+1, είτε υποχωρεί, είτε παραμένει στη θέση του. Αν d η πρόταση που έκανε ο ίδιος στον προηγούμενο γύρο k και d1 η πρόταση του αντιπάλου, τότε:

```
if (getRisk(d,d1,myself) > getRisk(d1,d,opponent))   return(d);
else   return(findNextDeal());
```

Προκειμένου να επιτύχει στο νέο γύρο k+1 την ελάχιστη δυνατή υποχώρηση, μπορεί, να μη στείλει απευθείας την επόμενη πρόταση στον αντίπαλο, αλλά πρώτα να υπολογίσει αν αυτή θα αλλάξει την ισορροπία του ρίσκου, αναγκάζοντας στον επόμενο γύρο τον αντίπαλο σε υποχώρηση. Αν δεν αλλάζει την ισορροπία του ρίσκου, τότε μπορεί να δοκιμάσει την επόμενη υποψήφια πρόταση, κοκ, μέχρι να επιτευχθεί ο στόχος.

Για τον υπολογισμό της διαφοράς ρίσκου θα χρησιμοποιηθεί για τον αντίπαλο η παλιά πρόταση d1. Αυτό είναι ορθό στην περίπτωση που στο γύρο k+1 θα υποχωρήσει μόνο ο ίδιος ο πράκτορας, όχι και ο αντίπαλος, σύμφωνα με τη σχέση $risk^kM < risk^kO$ χωρίς την ισότητα (όχι ≤). Στην περίπτωση που υποχωρεί και ο αντίπαλος, δεν εφαρμόζουμε την προηγούμενη επαναληπτική διαδικασία. Η ισορροπία ρίσκου δεν εξαρτάται πλέον μόνο από την πρόταση του πράκτορα και μπορεί ευκολότερα να ανατραπεί.

Αλγόριθμος 3.11: Επίτευξη ελάχιστης δυνατής υποχώρησης

```
equalRiskDif = false;
riskDif = getRisk(d,d1,myself) - getRisk(d1,d,opponent);
if (riskDif  > 0)
    return(d);
if (riskDif == 0)
   equalRiskDif = true;
do {
   d' = findNextDeal();
   if (equalRiskDif || getRisk(d',d1,myself) >= getRisk(d1,d',opponent))
      return(d');
} while (true);
```

Απλοποίηση τύπων.

Στη συνέχεια, θα αποπειραθούμε να απλοποιήσουμε κάποιους συχνά χρησιμοποιούμενους τύπους. Χρησιμοποιώντας τους στην υλοποίηση, γίνονται λιγότερες πράξεις, δηλαδή έχουμε υπολογιστικό κέρδος.

- Για να κρίνει ένας πράκτορας αν τον συμφέρει να αποδεχτεί μια πρόταση, ελέγχει αν $u_M(\delta_O) > u_M(\delta_M) \Leftrightarrow c_M(T) - c_M(\delta_O) > c_M(T) - c_M(\delta_M)$.
  Οπότε $u_M(\delta_O) > u_M(\delta_M) \Leftrightarrow c_M(\delta_O) < c_M(\delta_M)$.
  Γλιτώνουμε 2 πράξεις.
- Ειδικά για τις απλές εργασίες
  Για να κρίνει ένας πράκτορας που χρησιμοποιεί τη στρατηγική zeuthen αν τον συμφέρει να υποχωρήσει σε ένα βήμα, ελέγχει αν:
  $r_M < r_O \Leftrightarrow (u_M(\delta_M) - u_M(\delta_O)) / u_M(\delta_M) < (u_O(\delta_O) - u_O(\delta_M)) / u_O(\delta_O)$ (I).
  Όμως $u_M(\delta_M) - u_M(\delta_O) = c_M(T) - c_M(\delta_M) - c_M(T) + c_M(\delta_O) = c_M(\delta_O) - c_M(\delta_M)$ (II).
  ισχύει $c_M(\delta_O) = totalCost - c_O(\delta_O)$ (III).
  Από τις (II) και (III), $u_M(\delta_M) - u_M(\delta_O) = totalCost - c_O(\delta_O) - totalCost + c_O(\delta_M) =$
  $= c_O(T) - c_O(\delta_O) - c_O(T) + c_O(\delta_M) = u_O(\delta_O) - u_O(\delta_M)$ (IV).
  Από τις (I) και (VI), $r_M < r_O \Leftrightarrow 1 / u_M(\delta_M) < 1 / u_O(\delta_O)$.
  Οπότε, $r_M < r_O \Leftrightarrow u_M(\delta_M) > u_O(\delta_O)$.
  Ισοδύναμα, $r_M < r_O \Leftrightarrow c_M(T) - c_M(\delta_M) > c_O(T) - c_O(\delta_O) \Leftrightarrow$
  $\Leftrightarrow c_M(T) - c_M(\delta_M) > c_O(T) - (totalCost - c_M(\delta_O)) \Leftrightarrow$

$\Leftrightarrow c_M(\delta_M) + c_M(\delta_O) < c_M(T) - c_O(T) + totalCost.$
Οπότε, $r_M < r_O \Leftrightarrow c_M(\delta_M) + c_M(\delta_O) < z$,
όπου $z = c_M(T) - c_O(T) + totalCost$ *σταθερά.*
Γλιτώνουμε 7 πράξεις.

Αναπαράσταση υποσυνόλων εργασιών.

Στο σημείο αυτό πρέπει να αναλύσουμε καλύτερα τον τρόπο αναπαράστασης ενός υποσυνόλου που αναφέρθηκε σε προηγούμενο υποκεφάλαιο. Κάθε εργασία αντιστοιχεί σε ένα από N δυαδικά ψηφία και μάλιστα η t_{N-1} αντιστοιχεί στο πιο σημαντικό ψηφίο. Η ύπαρξη μιας εργασίας σε ένα υποσύνολο αντιστοιχεί στο ψηφίο 1 και η απουσία της σε 0. Ο δυαδικός αριθμός N ψηφίων μετατρέπεται στον αντίστοιχο δεκαδικό ακέραιο. Για παράδειγμα για 4 εργασίες ισχύει $\{t_0,t_1,t_2,t_3\} \leftrightarrow 1111 \leftrightarrow 15$, $\{t_0,t_1,t_2\} \leftrightarrow 0111 \leftrightarrow 7$, $\{t_1\} \leftrightarrow 0010 \leftrightarrow 2$, κοκ.

Αλγόριθμος 3.12: Μετατροπή συμβολικού ακεραίου subset σε πρόταση d

```
boolean[][] subsetToDeal(int subset) {
    for (i=0; i<N; i++)
        if ((subset & 2i) != 0)
            d[myself][i] = true;
        else
            d[myself][i] = false;
    d = getContemporary(d); // από αλγόριθμο 3.3
    return d;
}
```

Λεπτομέρειες υλοποίησης μεθόδων.

Αναφορικά με την υλοποίηση των μεθόδων ΕΜΑ και ΠΜΨ, οι λίστες υποεργασιών αναπαρίστανται με πίνακες. Συγκεκριμένα, για απλές εργασίες ορίζεται ένας πίνακας subsets int[2][2^N]. Η πρώτη γραμμή του πίνακα περιέχει το υποσύνολο εργασιών και η δεύτερη το αντίστοιχο κόστος πράκτορα. Για σύνθετες εργασίες, ο πίνακας περιέχει μια ακόμη γραμμή με το κόστος αντιπάλου. Η συμπλήρωση των κελιών γίνεται ως εξής:

subsets[0][0] = $0 \leftrightarrow \{\}$, subsets[1][0] = 0

0. Η επόμενη στήλη προκύπτει προσθέτοντας στην προηγούμενη $2^0 \leftrightarrow t_0$ και c_0
   subsets[0][1]=subsets[0][0]+2^0=$1 \leftrightarrow \{t_0\}$, subsets[1][1]=subsets[1][0]+c_0=c_0
1. Οι επόμενες 2 στήλες προκύπτουν προσθέτοντας στις προηγούμενες τα $2^1 \leftrightarrow t_1$ και c_1
   subsets[0][2]=subsets[0][0]+2^1=$2 \leftrightarrow \{t_1\}$, subsets[1][2]=subsets[1][0]+c_1=c_1

subsets[0][3] = subsets[0][1]+2^1 = 3↔{t_0,t_1},

subsets[1][3] = subsets[1][1]+c_1 = c_0+c_1 …

i. Οι επόμενες 2^i στήλες προκύπτουν προσθέτοντας στις προηγούμενες τα 2^i↔t_i και c_i

```
for (j = 2^i to 2^(i+1)-1) {
    subsets[0][j] = subsets[0][j-2^i] + 2^i
    subsets[1][j] = subsets[1][j-2^i] + c_i
}
```

Για σύνθετες εργασίες πρέπει να ελεγχθεί αν το υποσύνολο που θα προκύψει είναι αποδεκτό και να υπολογιστεί επιπλέον το κόστος αντιπάλου:

```
for (j = 2^i to 2^(i+1)-1)
    if (isValidChoice(i,subsets[0][j-2^i])) {
        subsets[0][j] = subsets[0][j-2^i] + 2^i
        subsets[1][j] = subsets[1][j-2^i] + c_i
        c´_i = calculateSubtaskCost(i,subsets[0][j-2^i])
        subsets[2][j] = subsets[2][j-2^i] - c´_i
    }
```

Για την υλοποίηση των mergeSubsets και trimFinalList που είδαμε στο προηγούμενο κεφάλαιο για απλές και σύνθετες εργασίες με τις μεθόδους ΕΜΑ και ΠΜΨ δεν έχουμε να σχολιάσουμε κάτι.

Μετά τον offline υπολογισμό των υποσυνόλων εργασιών και του αντίστοιχου κόστους ξεκινάει η διαδικασία διαπραγμάτευσης. Σε κάθε γύρο η επόμενη πρόταση προκύπτει από τη μετατροπή του subsets[0][currentPos], όπως είδαμε προηγουμένως. Η μετατροπή έχει πολυπλοκότητα O(N).

Αναφορικά με τη μέθοδο ΨΜΑ, χρησιμοποιούνται οι πίνακες subsets = int[N+1][W+1], ccM = int[N+1][W+1] και ccO = int[N+1][W+1]. Στον subsets αποθηκεύονται τα υποσύνολα εργασιών ως αριθμοί με τον τρόπο που μόλις περιγράψαμε.

Τέλος, για τη μέθοδο ΕΜ αποθηκεύεται κάθε φορά μόνο η τρέχουσα πρόταση και κόστος για τον πράκτορα και τον αντίπαλο και δεν κρατάμε παλιότερα αποτελέσματα.

3.4. Διασύνδεση με το χρήστη

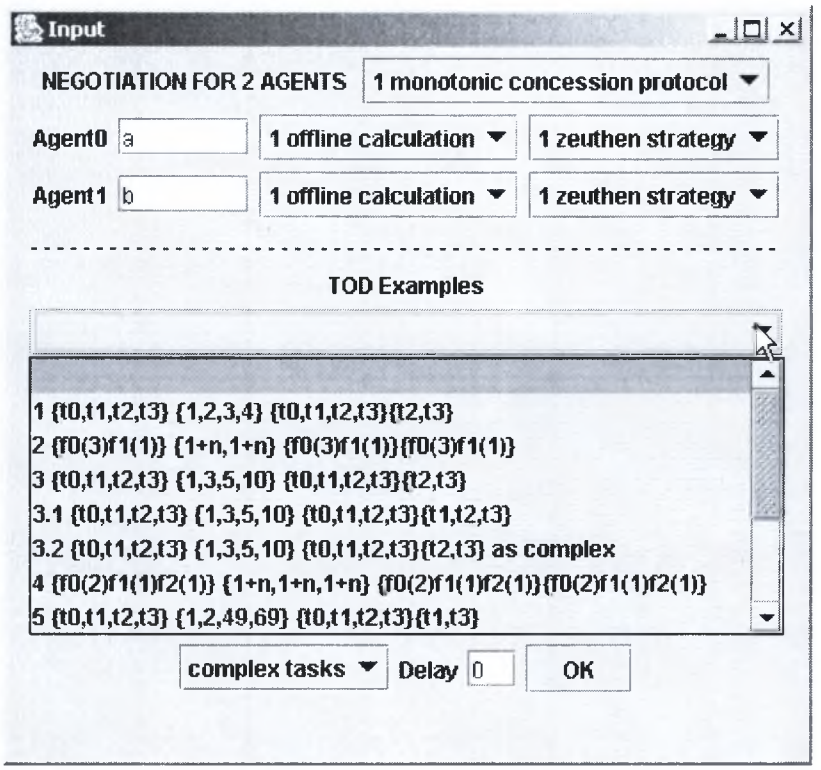
Είσοδος στοιχείων στο σύστημα.

Στην αρχή της εκτέλεσης, προβάλει μια οθόνη, για την είσοδο στοιχείων στο πρόγραμμα, που υλοποιείται από την κλάση Input.

Εδώ εισάγονται το χρησιμοποιούμενο πρωτόκολλο, τα ονόματα των δυο πρακτόρων, η μέθοδος και η πολιτική υποχώρησης που χρησιμοποιεί καθένας.

Χρησιμοποιούμενες μέθοδοι	
ΕΜΑ	1 offline calculation
ΨΜΑ	2 offline with DP
ΠΜΨ	3 trim lists by ε
ΕΜ	4 heuristic method

Χρησιμοποιούμενες πολιτικές	
στρατηγική Zeuthen	1 zeuthen strategy
συνεχής υποχώρηση	2 always concede
μηδενική υποχώρηση	3 never concede
τυχαία υποχώρηση	4 random concede



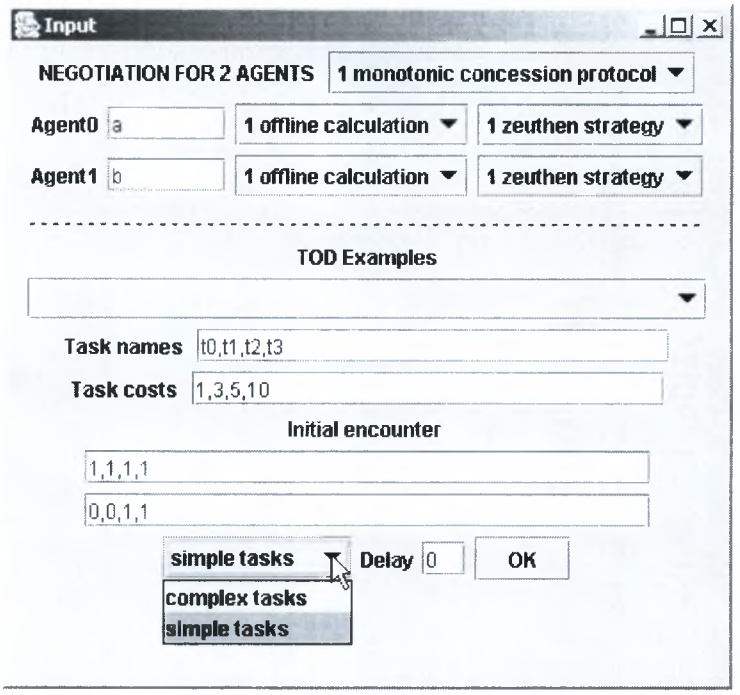
Αν επιλεγεί η μέθοδος ΠΜΨ τότε καταχωρείται και ο παράγοντας ε, ενώ αν επιλεγεί η πολιτική 4 τυχαίας υποχώρησης εισάγεται και το αντίστοιχο ποσοστό.

Αναφορικά με τις εργασίες, μπορεί να χρησιμοποιηθεί κάποιο από τα έτοιμα παραδείγματα. Εναλλακτικά, μπορούν να πληκτρολογηθούν, χωρισμένα από κόμματα, τα ονόματα των εργασιών, το κόστος των εργασιών και η αρχική αλληλεπίδραση. Για τις σύνθετες εργασίες το κόστος καταχωρείται σε δυο πλαίσια. Αν το κόστος είναι τις μορφής c0+c1*n, τότε στο πρώτο πλαίσιο καταχωρούνται τα c0 χωρισμένα από κόμματα και στο δεύτερο τα c1.

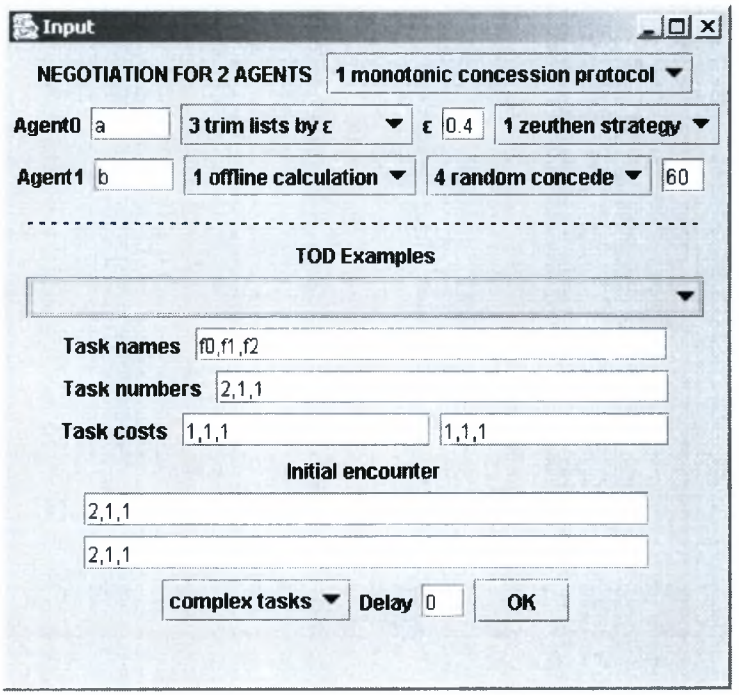
Η αρχική αλληλεπίδραση καταχωρείται σε δυο πεδία ως εξής: στο πρώτο μπαίνει το πλήθος υποεργασιών κάθε εργασίας που αντιστοιχούν στον πράκτορα 0 και στο δεύτερο το αντίστοιχο πλήθος για τον πράκτορα 1. Για τις απλές εργασίες, κατά ανάλογο τρόπο, μπαίνει 1 αν η εργασία ανατίθεται στον

πράκτορα και 0 αλλιώς. Σε περίπτωση που τα πεδία μείνουν κενά, τότε η αρχική αλληλεπίδραση ορίζεται τυχαία από το πρόγραμμα.

Παράδειγμα. Έστω δυο πράκτορες a και b και έστω οι εργασίες $\{t_0, t_1, t_2, t_3\}$ με κόστος $\{1, 3, 5, 10\}$ και αρχική αλληλεπίδραση $<\{t_0, t_1, t_2, t_3\}, \{t_2, t_3\}>$.



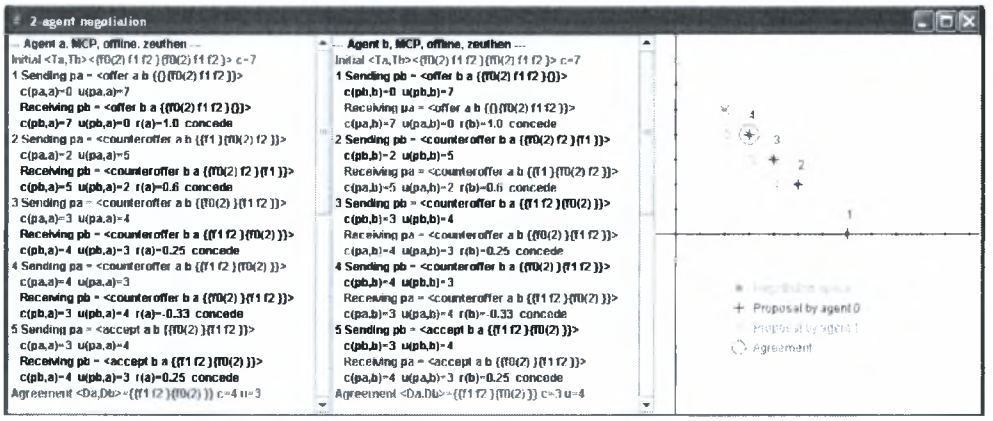
Παράδειγμα. Έστω δυο πράκτορες a και b και έστω οι εργασίες $\{f_0(2), f_1(1), f_2(1)\}$. Οι συναρτήσεις κόστους είναι: $c_0(n)=1+n$, $c_1(n)=1+n$, $c_2(n)=1+n$. Αρχική αλληλεπίδραση $<\{f_0(2), f_1(1), f_2(1)\}, \{f_0(2), f_1(1), f_2(1)\}>$.

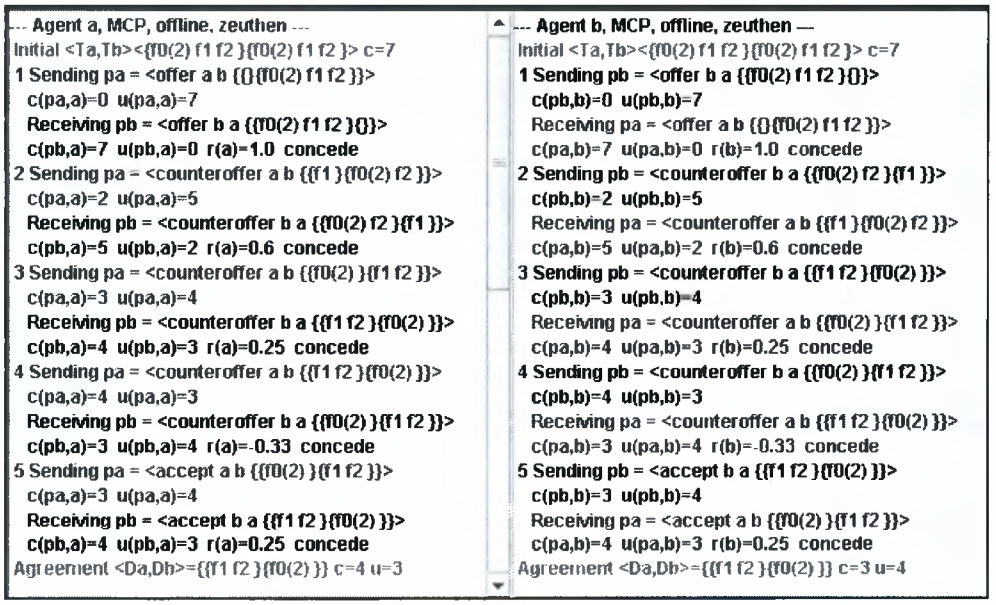


Έξοδος αποτελεσμάτων.

Η πορεία της διαπραγμάτευσης μεταξύ των πρακτόρων φαίνεται μέσα από τα μηνύματα και τη γραφική παράσταση που εμφανίζονται σε ειδικό παράθυρο της οθόνης με τη βοήθεια της κλάσης Output.

Για το ΜΠΣ εμφανίζεται μια εικόνα όπως η παρακάτω:





Στην πρώτη γραμμή εμφανίζεται το όνομα του πράκτορα, το πρωτόκολλο, η μέθοδος και η στρατηγική που χρησιμοποιεί. Στην επόμενη γραμμή, η αρχική αλληλεπίδραση και το κόστος για τον πράκτορα. Στη συνέχεια, για κάθε βήμα γράφεται η πρόταση που στάλθηκε και αυτή που παραλήφθηκε. Αναγράφεται για κάθε μια το κόστος και η χρησιμότητα για τον πράκτορα και, στην περίπτωση που χρησιμοποιείται Zeuthen, το υπολογιζόμενο ρίσκο. Στην τελευταία γραμμή, εμφανίζεται το αποτέλεσμα, δηλαδή αν επιτεύχθηκε απόρριψη ή συμφωνία, και ποια είναι αυτή.

Επίσης, απεικονίζεται το σύνολο διαπραγμάτευσης και οι προτάσεις κάθε πράκτορα στο χώρο. Ο άξονας x αντιστοιχεί στη χρησιμότητα του πράκτορα a και ο άξονας y στη χρησιμότητα του b. Πριν ξεκινήσει η διαδικασία, υπολογίζονται με τη βοήθεια του offline υπολογισμού τα σημεία του συνόλου διαπραγμάτευσης ως ζεύγη της μορφής (χρησιμότητα πράκτορα a, χρησιμότητα πράκτορα b) και εμφανίζονται ως πράσινοι μικροί κύκλοι. Σε κάθε βήμα της διαπραγμάτευσης, μαρκάρεται με ένα μπλε + η πρόταση του πράκτορα a και με ένα γκρι x η πρόταση του πράκτορα b. Τέλος, αν υπάρξει συμφωνία, σημειώνεται με ένα κόκκινο κύκλο.

Παρατήρηση.

Στις απλές εργασίες τα σημεία του συνόλου διαπραγμάτευσης, βρίσκονται πάντα πάνω σε μια ευθεία. Αυτό αποδεικνύεται ως εξής: Ισχύει cost(d,O) = totalCost - $cost_M$(d). Όμως, $utility_M$(d) = $cost_M$(e) - $cost_M$(d) και $utility_O$(d) = $cost_O$(e) - $cost_O$(d).

Άρα, $cost_O$(e) - $utility_O$(d) = totalCost – $cost_M$(e) + $utility_M$(d) ⇒

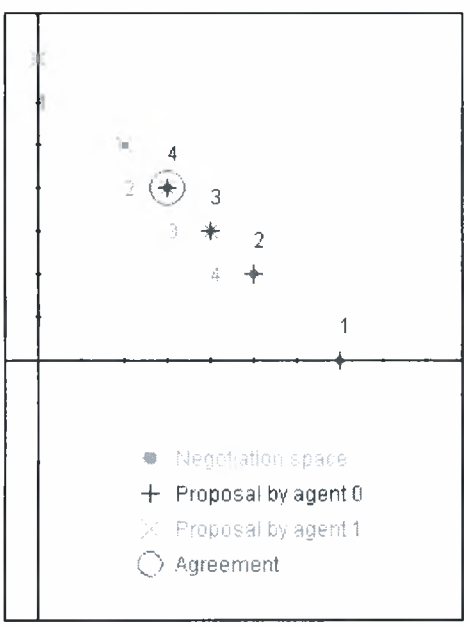
$utility_O$(d) = $cost_M$(e) + $cost_O$(e) – totalCost – $utility_M$(d) ⇒

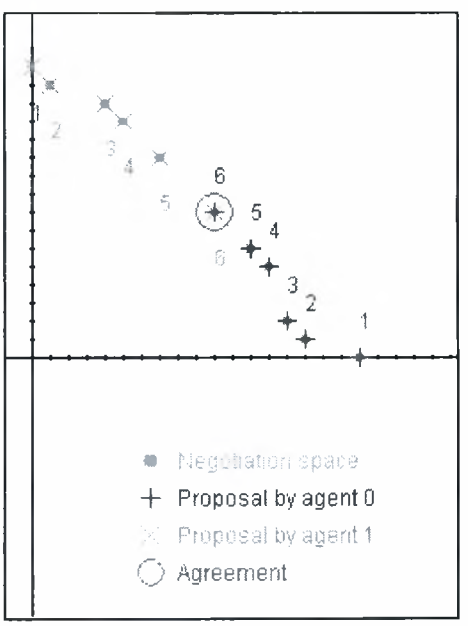
$utility_O$(d) = σταθερά – $utility_M$(d).

Στις σύνθετες εργασίες το παραπάνω δεν ισχύει πάντα.

Για παράδειγμα, έστω {f0(3), f1(2), f2(1)} με κόστος c0=2+n, c1=2+2n, c2=5+3n.

Αρχική αλληλεπίδραση {{f0(2), f1(2), f2(1)}, {f0(3), f1(2), f2(1)}. Παρατηρούμε ότι πλέον τα σημεία του συνόλου διαπραγμάτευσης δεν βρίσκονται πάνω σε ευθεία, αλλά σε καμπύλη.





Ανακεφαλαίωση

Στο κεφάλαιο 3, περιγράφηκε η αρχιτεκτονική του συστήματος και πραγματοποιήθηκε αντικειμενοστραφής ανάλυση. Στη συνέχεια, έγιναν κάποιες συμβάσεις και δόθηκαν λεπτομέρειες για τη σχεδίαση και την υλοποίηση του συστήματος. Ιδιαίτερη έμφαση δόθηκε στις εξηγήσεις για τον τρόπο αναπαράστασης των πρακτόρων, των απλών και σύνθετων εργασιών, των προτάσεων και αλληλεπιδράσεων. Τέλος, εξηγήθηκαν οι οθόνες διασύνδεσης με το χρήστη για είσοδο και έξοδο στοιχείων.

Στο επόμενο κεφάλαιο, θα μελετηθούν με τη βοήθεια του συστήματος διαφορετικά σενάρια χρήσης.

4. Σενάρια εκτέλεσης.

Στο κεφάλαιο αυτό θα προσπαθήσουμε να επαληθεύσουμε πειραματικά τα συμπεράσματα του προηγούμενου κεφαλαίου, αναφορικά με τις μεθόδους και πολιτικές που χρησιμοποιούν οι πράκτορες. Σε όλες τις περιπτώσεις θεωρούμε ότι έχουμε 2 πράκτορες {a, b}, που είναι ειλικρινείς, δηλαδή δεν προσπαθεί ο ένας να εκμεταλλευτεί τον άλλο. Θα μελετήσουμε διάφορα σενάρια με απλές και σύνθετες εργασίες και πιο συγκεκριμένα τα παρακάτω.

Σενάριο 1. Έστω οι απλές εργασίες $\{t_0, t_1, t_2, t_3\}$ με κόστος {1, 3, 5, 10} και αρχική αλληλεπίδραση $e = <\{t_0, t_1, t_2, t_3\}\ \{t_1, t_2, t_3\}>$. Ισχύει $c_a(e) = 19$ και $c_b(e) = 18$. Αρχικό σημείο (a) = 19–18 = 1 και αρχικό (b) = 19–19 = 0. Τελικό (a) = 19 και τελικό (b) = 18.

Σενάριο 1.1. Έστω οι απλές εργασίες $\{t_0, t_1, t_2, t_3\}$ με κόστος {1, 3, 5, 10} και αρχική αλληλεπίδραση $e = <\{t_0, t_1, t_2, t_3\}\ \{t_2, t_3\}>$. $c_a(e) = 19$ και $c_b(e) = 15$. Αρχικό σημείο (a) = 19–15 = 4 και αρχικό (b) = 19–19 = 0. Τελικό (a) = 19 και τελικό (b) = 15.

Σενάριο 1.2. Έστω οι απλές εργασίες {t0, t1, t2, t3} με κόστος {1, 3, 5, 20} και αρχική αλληλεπίδραση $e = <\{t_1, t_2, t_3\}\ \{t_0, t_1, t_3\}>$. $c_a(e) = 28$ και $c_b(e) = 24$. Αρχικό σημείο (a) = 29–24 = 5 και αρχικό (b) = 29–28 = 1. Τελικό (a) = 28 και τελικό (b) = 24.

Σενάριο 2. Έστω οι σύνθετες εργασίες $\{f_0(2), f_1(1), f_2(1)\}$ με συναρτήσεις κόστους $c_0(n)=1+n$, $c_1(n)=1+n$, $c_2(n)=1+n$. Αρχική αλληλεπίδραση $e = <\{f_0(2), f_1(1), f_2(1)\}\ \{f_0(2), f_1(1), f_2(1)\}>$. Ισχύει $c_a(e) = 7$ και $c_b(e) = 7$. Αρχικό σημείο (a) = 7–7 = 0 και αρχικό (b) = 7–7 = 0. Τελικό (a) = 7 και τελικό (b) = 7. Οι σύνθετες εργασίες αναλύονται στις υποεργασίες $\{f_0, f_0, f_1, f_2\}$. Εσωτερικά στο σύστημα δημιουργούνται οι πίνακες costI = {2, 1, 2, 2} και dependency = {-1, 0, -1, -1}.

Σενάριο 3. Έστω οι απλές εργασίες $\{t_0, t_1, t_2, t_3\}$ με κόστος {1, 2, 49, 69} και αρχική αλληλεπίδραση $e = <\{t_0, t_1, t_2, t_3\}\ \{t_1, t_3\}>$. Ισχύει $c_a(e) = 121$ και $c_b(e) = 71$. Αρχικό σημείο (a) = 121–71 = 50 και αρχικό (b) = 121–121 = 0. Τελικό (a) = 121 και τελικό (b) = 71.

Σενάριο 3.1. Έστω οι απλές εργασίες $\{t_0, t_1, t_2, t_3\}$ με κόστος {1, 2, 49, 69} και αρχική αλληλεπίδραση $e = <\{t_1, t_2, t_3\}\ \{t_0, t_1, t_2\}>$. Ισχύει $c_a(e) = 120$ και $c_b(e) = 52$. Αρχικό σημείο (a) = 121–52 = 69 και αρχικό (b) = 121–120 = 1. Τελικό (a) = 120 και τελικό (b) = 52.

Σενάριο 4. Έστω οι σύνθετες εργασίες $\{f_0(3), f_1(1)\}$ με συναρτήσεις κόστους $c_0(n) = 1+n$ και $c_1(n) = 1+n$. Αρχική αλληλεπίδραση $e = <\{f_0(3), f_1(1)\}\ \{f_0(3), f_1(1)\}>$. Ισχύει $c_a(e) = 6$ και $c_b(e) = 6$. Αρχικό σημείο (a) = 6–6 = 0 και αρχικό (b) = 6–0 = 6. Τελικό (a) = 0 και τελικό (b) = 6. Οι σύνθετες εργασίες αναλύονται στις υποεργασίες {f0, f0, f0, f1}. Εσωτερικά στο σύστημα δημιουργούνται οι πίνακες costI = {2, 1, 1, 2} και dependency = {-1, 0, 0, -1}.

Σενάριο 5. Έστω οι σύνθετες εργασίες $\{f_0(3), f_1(2), f_2(1)\}$ με συναρτήσεις κόστους $c_0(n)=2+n$, $c_1(n)=2+2n$, $c_2(n)=5+3n$. Αρχική αλληλεπίδραση $e = <\{f_0(2), f_1(2), f_2(1)\}\ \{f_0(3), f_1(2), f_2(1)\}>$. Ισχύει $c_a(e) = 18$ και $c_b(e) = 19$. Αρχικό σημείο (a) = 19–19 = 0 και αρχικό (b) = 19–18 = 1. Τελικό (a) = 18 και τελικό (b) = 19. Οι σύνθετες εργασίες αναλύονται στις υποεργασίες $\{f_0, f_0, f_0, f_1, f_1, f_2\}$. Εσωτερικά στο σύστημα δημιουργούνται οι πίνακες costI = {3, 1, 1, 4, 2, 8} και dependency = {-1, 0, 0, -1, 3, -1}.

Σενάριο 6. Έστω οι εργασίες $\{t_0, t_1, f_2(2)\}$ με κόστος $c_0=3$, $c_1=4$, $c_2(n)=2+2n$ και αρχική αλληλεπίδραση $e = <\{t_0, t_1, f_2(1)\}\ \{f_2(2)\}>$. Ισχύει $c_a(e) = 11$ και $c_b(e) = 6$. Αρχικό σημείο (a) = 13–6 = 7 και αρχικό (b) = 13–11 = 2. Τελικό (a) = 11 και τελικό (b) = 6. Τις αντιμετωπίζουμε ως $\{t_0(1), t_1(1), f_2(1), f_2(1)\}$ ή απλούστερα $\{t_0, t_1, f_2, f_2\}$. Εσωτερικά στο σύστημα η πληροφορία αναπαρίσταται ως: taskCost = {{3,4,2}, {0,0,2}}, taskNumber = {1,1,2}, ie = {{1,1,1}, {0,0,2}}, costI = {3,4,4,2} και dependency = {-1,-1,-1,2}.

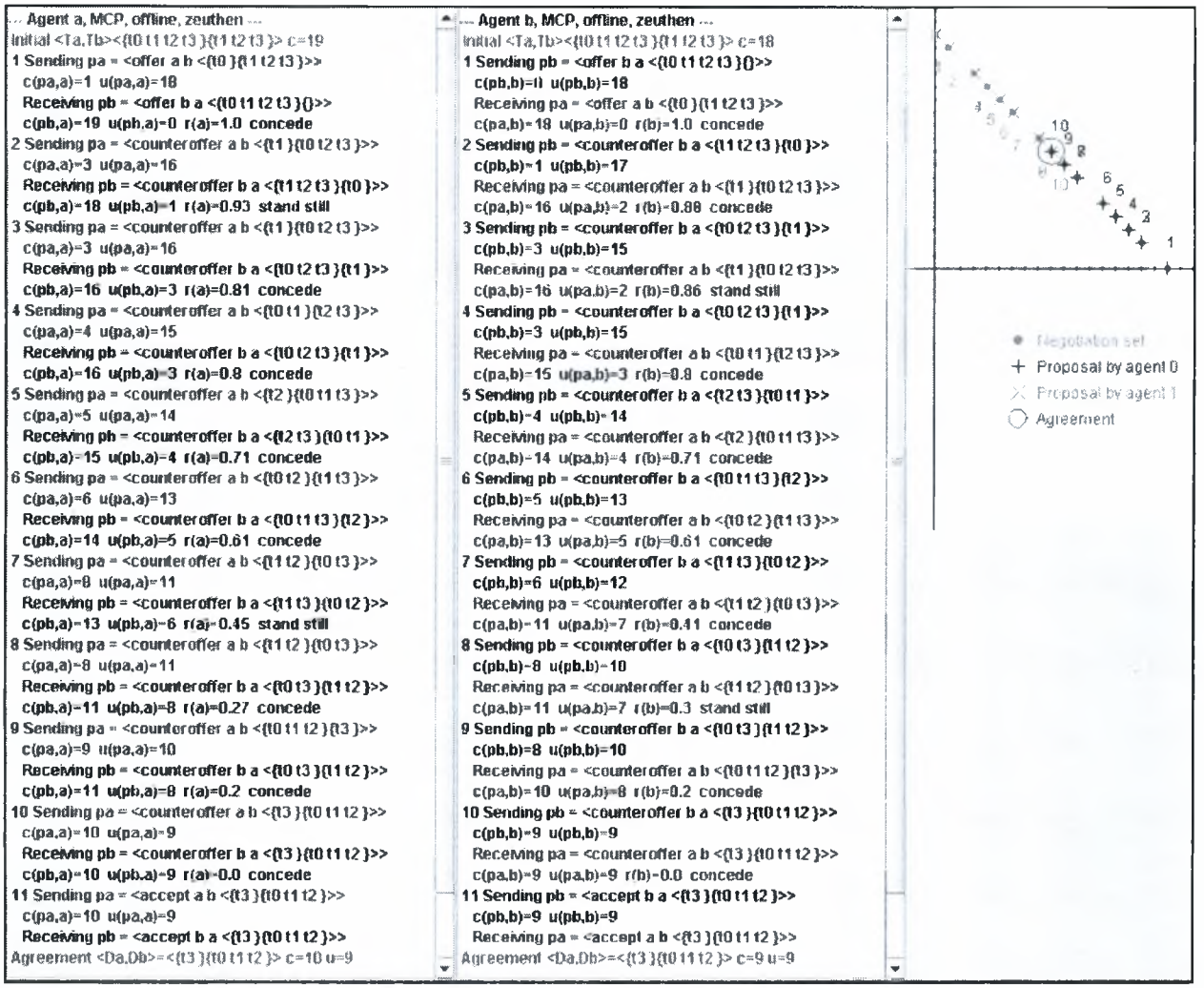
4.1. Σύγκριση μεθόδων εύρεσης επόμενης πρότασης.

Για να συγκρίνουμε τις διάφορες μεθόδους μεταξύ τους, θα θεωρήσουμε ότι και οι δυο πράκτορες εφαρμόζουν τη στρατηγική Zeuthen.

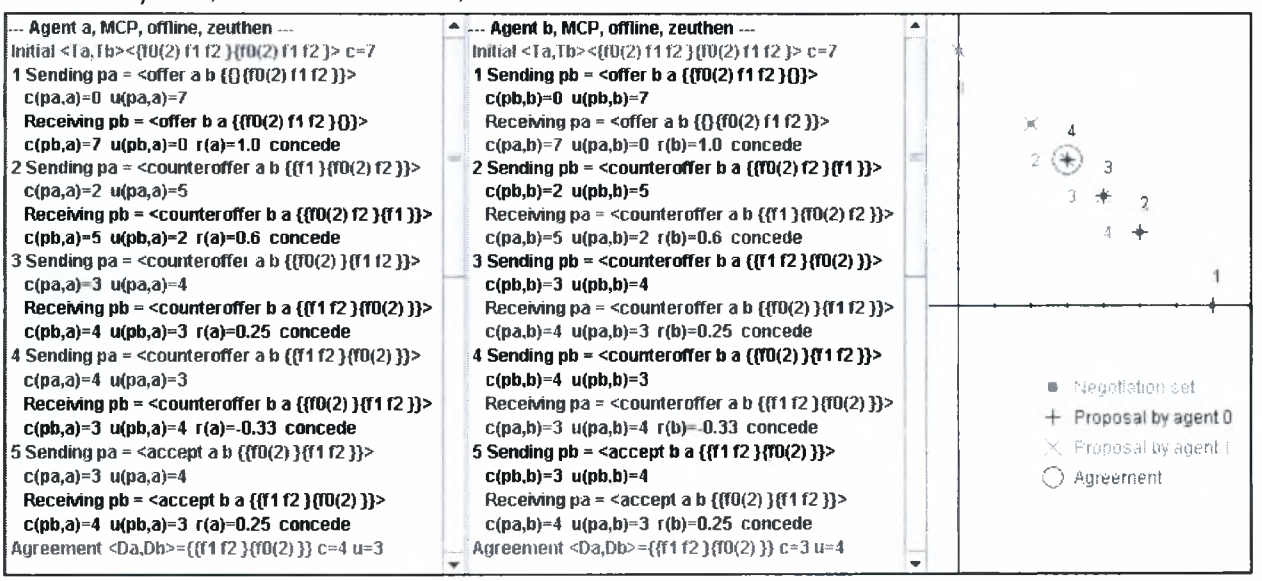
Εκθετική Μέθοδος Ακρίβειας και Ψευδοπολυωνυμική Μέθοδος Ακρίβειας.

Καταρχήν, τόσο η ΕΜΑ όσο και η ΨΜΑ υπολογίζουν offline όλα τα υποσύνολα του συνόλου διαπραγμάτευσης. Έτσι παρουσιάζουν μεταξύ τους τα ίδια αποτελέσματα σε κάθε γύρο. Το τελικό αποτέλεσμα είναι επίσης το ίδιο, εκτός από την περίπτωση που στο τελευταίο βήμα αποδέχονται και οι δυο πράκτορες την πρόταση του αντιπάλου και γίνεται τυχαία επιλογή μιας από τις δύο. Το συμπέρασμα αυτό επαληθεύεται στη συνέχεια τα σενάρια 1 για απλές εργασίες και 2 για σύνθετες.

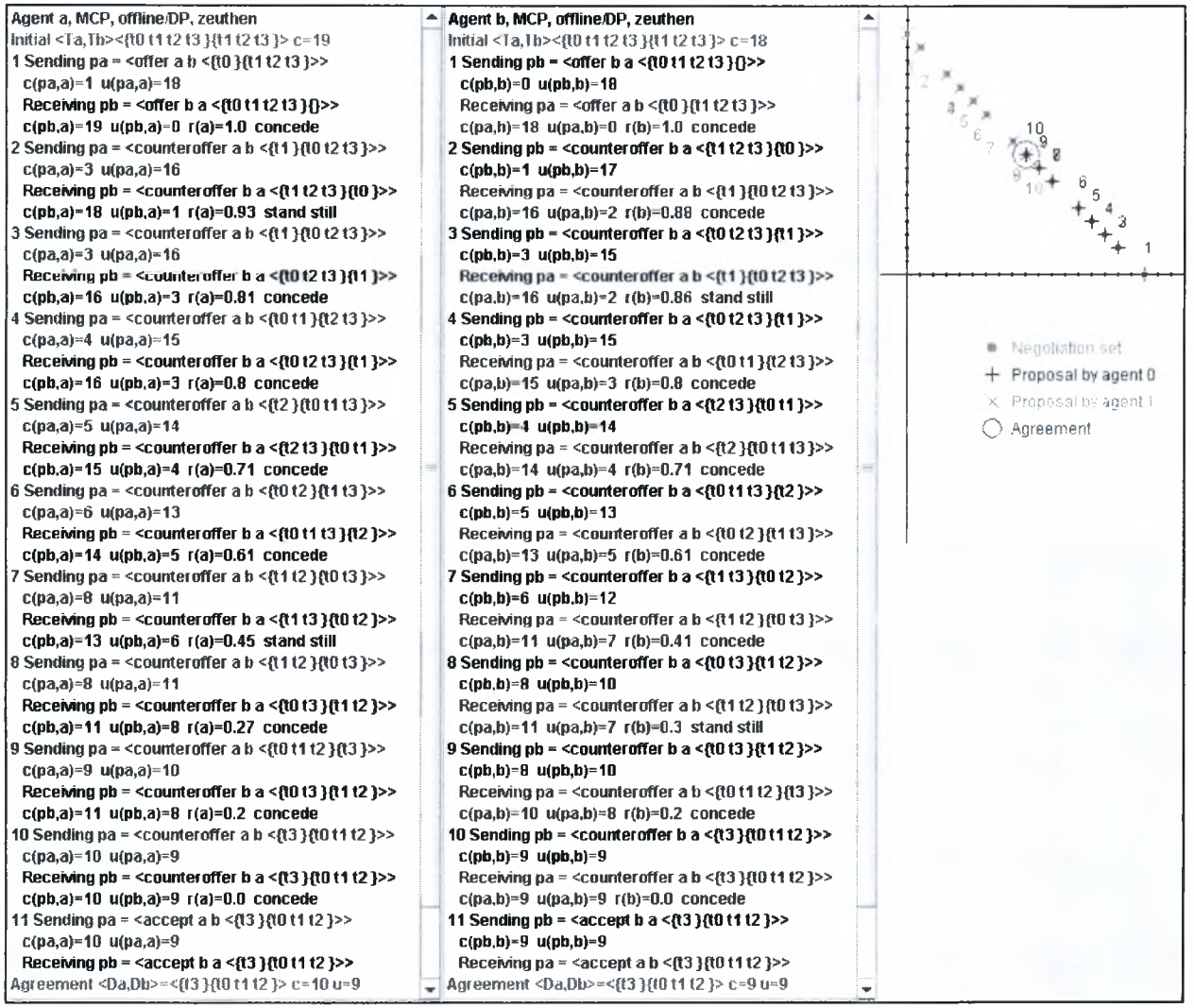
Σενάριο 1, ΜΠΣ & Zeuthen, ΕΜΑ.



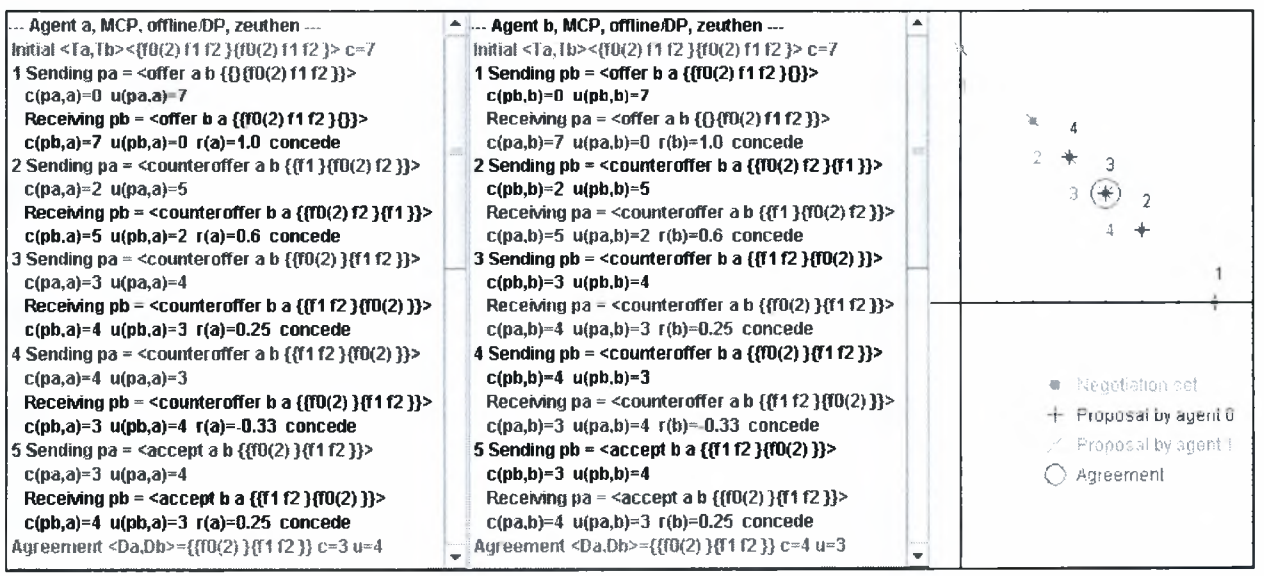
Σενάριο 2, ΜΠΣ & Zeuthen, ΕΜΑ.



Σενάριο 1, ΜΠΣ & Zeuthen, ΨΜΑ.



Σενάριο 2, ΜΠΣ & Zeuthen, ΨΜΑ.



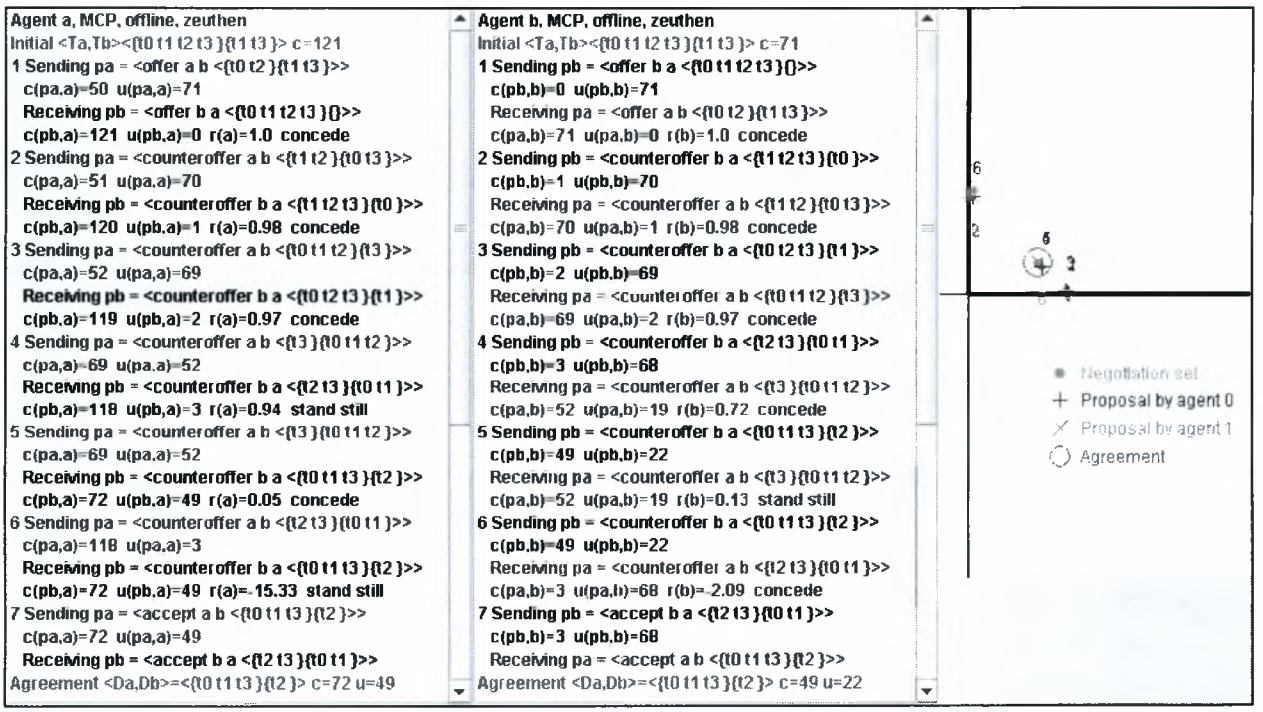
Πολυωνυμική Μέθοδος Ψαλιδίσματος.

Είναι δυνατό η διαπραγμάτευση με χρήση της ΠΜΨ να δώσει βέλτιστο αποτέλεσμα, ίδιο ακριβώς με αυτό της μεθόδου ΕΜΑ, και μάλιστα σε λιγότερα βήματα.

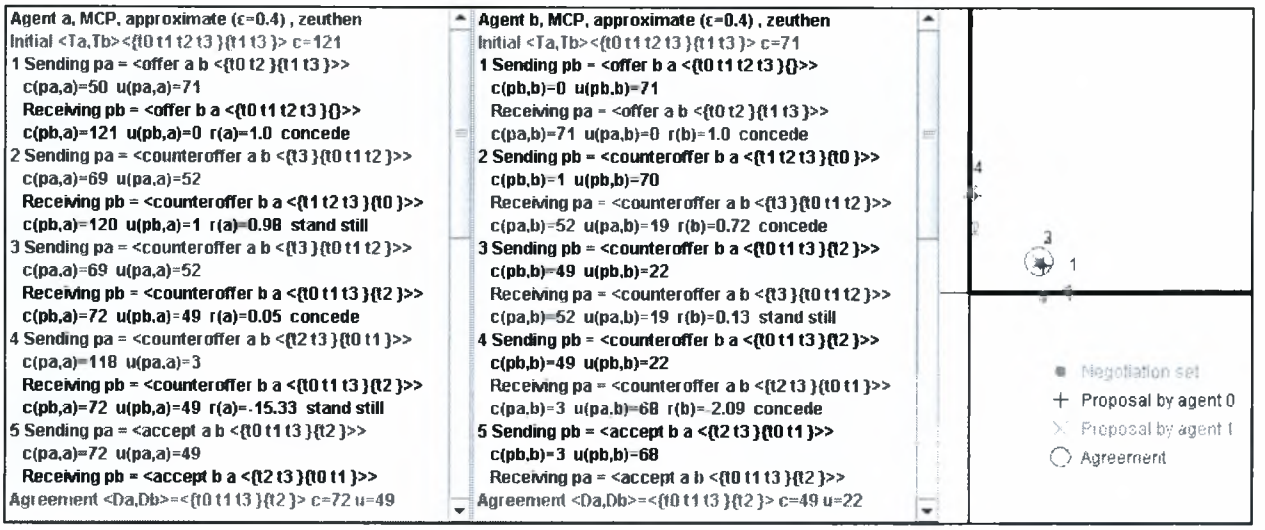
Στο σενάριο 3 για τις ΕΜΑ και ΨΜΑ το σύνολο διαπραγμάτευσης είναι για τον πράκτορα a {{t_0,t_2}-50, {t_1,t_2}-51, {t_0,t_1,t_2}-52, {t_3}-69, {t_0,t_3}-70, {t_1,t_3}-71, {t_0,t_1,t_3}-72, {t_2,t_3}-118, {t_0,t_2,t_3}-119, {t_1,t_2,t_3}-120, {t_0,t_1,t_2,t_3}-121} και για τον b {{}-0, {t_0}-1, {t_1}-2, {t_0,t_1}-3, {t_2}-49, {t_0,t_2}-50, {t_1,t_2}-51, {t_0,t_1,t_2}-52, {t_3}-69, {t_0,t_3}-70, {t_1,t_3}-71}.

Για την ΠΜΨ με ε = 0.4, οπότε δ = 0.4/8 = 0.05, η λίστα υποσυνόλων υποψήφιων για πρόταση είναι για τον πράκτορα a {{t_0,t_2}-50, {t_3}-69, {t_2,t_3}-118} και για τον b {{}-0, {t_0}-1, {t_1}-2, {t_0,t_1}-3, {t_2}-49, {t_0,t_1,t_2}-52, {t_3}-69}. Για τον πράκτορα a διαγράφονται οι τιμές κόστους 51, 52 που αντιπροσωπεύονται από το 50, οι 70, 71, 72 από το 69 και οι 119, 120, 121 από το 118. Για τον b οι τιμές 50, 51 αντιπροσωπεύονται από το 49 και οι τιμές 70, 71 από το 69.

Σενάριο 3, ΜΠΣ & Zeuthen, ΕΜΑ & ΨΜΑ.

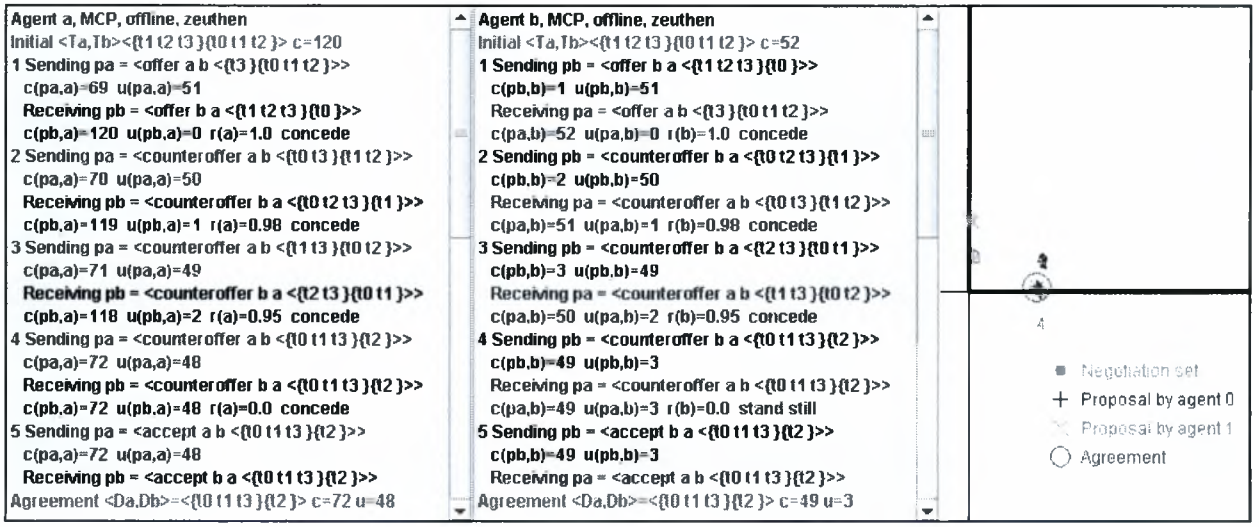


Σενάριο 3, ΜΠΣ & Zeuthen, ΠΜΨ.

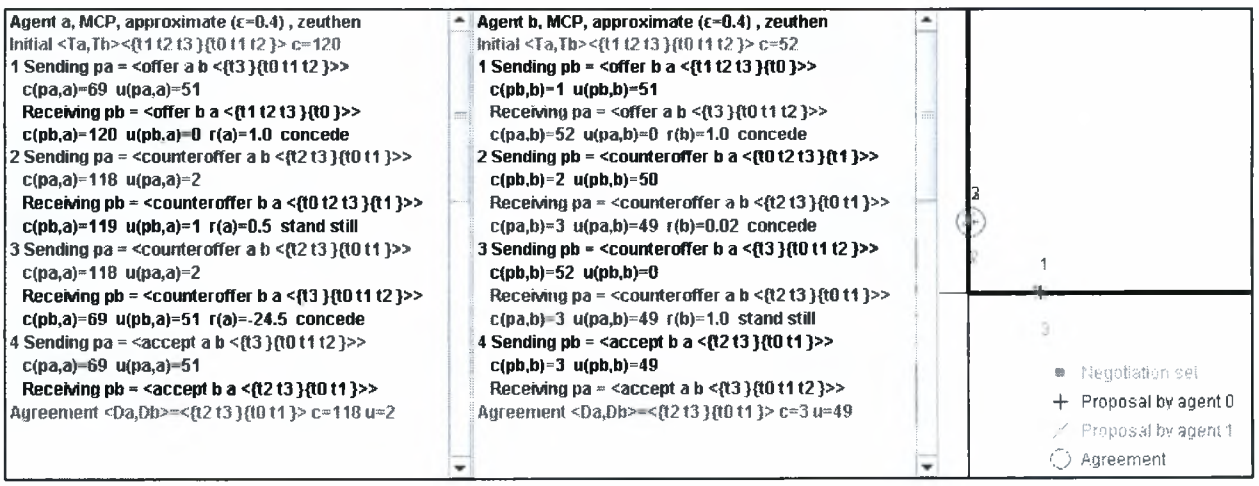


Είναι όμως δυνατό η διαπραγμάτευση με χρήση της ΠΜΨ να δώσει ένα αποτέλεσμα που απέχει αρκετά από τη βέλτιστη λύση, όπως φαίνεται στο σενάριο 3.1.

Σενάριο 3.1, ΜΠΣ & Zeuthen, ΕΜΑ & ΨΜΑ.

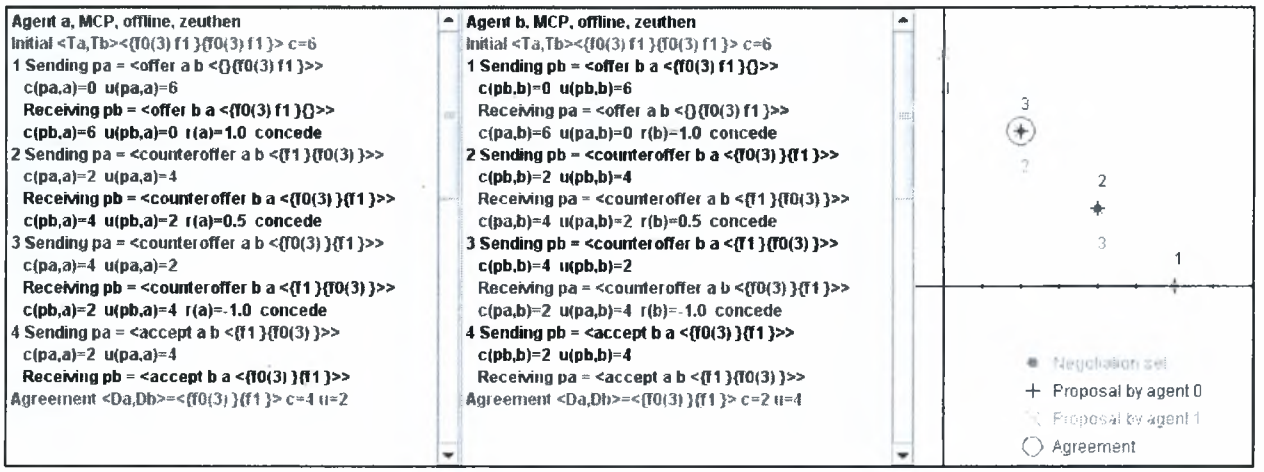


Σενάριο 3.1, ΜΠΣ & Zeuthen, ΠΜΨ.

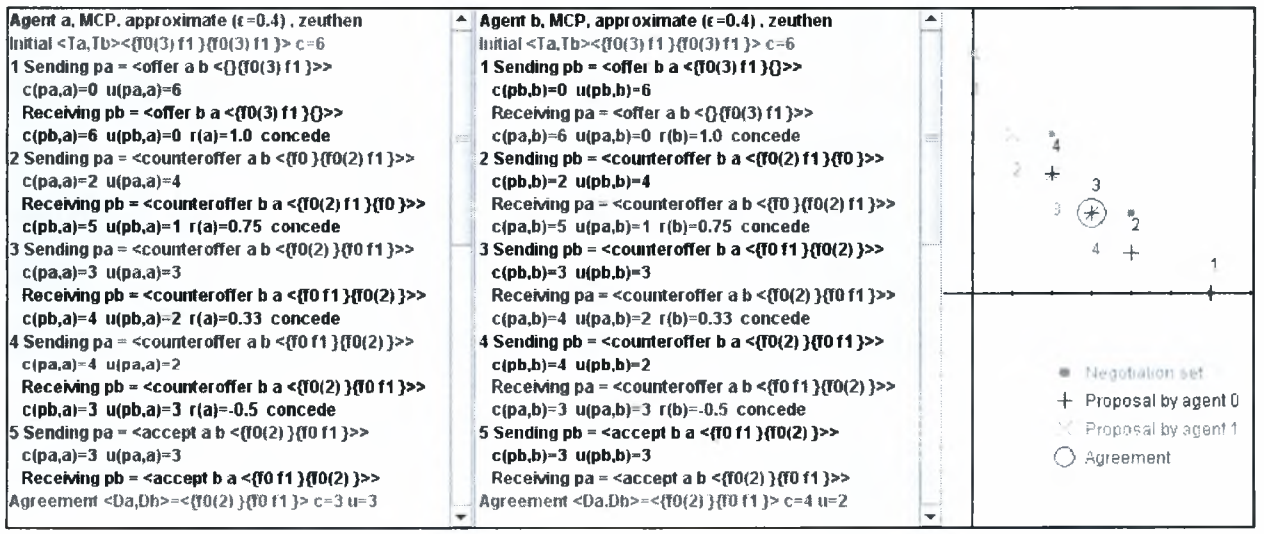


Με την ΠΜΨ είναι επίσης δυνατό οι προτάσεις των ενδιάμεσων ή και του τελικού γύρου της διαδικασίας να μην ανήκουν καν στο σύνολο διαπραγμάτευσης, γιατί δεν είναι pareto βέλτιστες. Αυτό επαληθεύεται και σχηματικά με το σενάριο 4, στο οποίο τα **x** και +, δηλαδή οι προτάσεις των πρακτόρων δεν ταυτίζονται με τις κουκίδες του συνόλου διαπραγμάτευσης.

Σενάριο 4, ΜΠΣ & Zeuthen, ΕΜΑ & ΨΜΑ.



Σενάριο 4, ΜΠΣ & Zeuthen, ΠΜΨ.

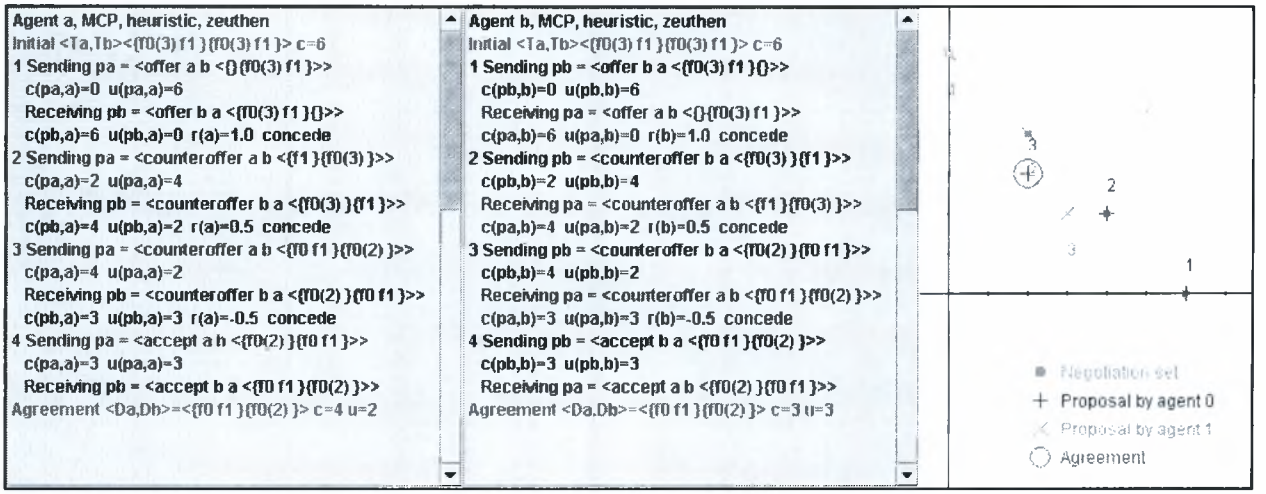


Ευρετική Μέθοδος.

Στη συνέχεια, θα ασχοληθούμε με την ΕΜ, με τη βοήθεια των παρακάτω σεναρίων στα οποία εφαρμόσαμε πριν τις μεθόδους ΕΜΑ, ΨΜΑ και ΠΜΨ.

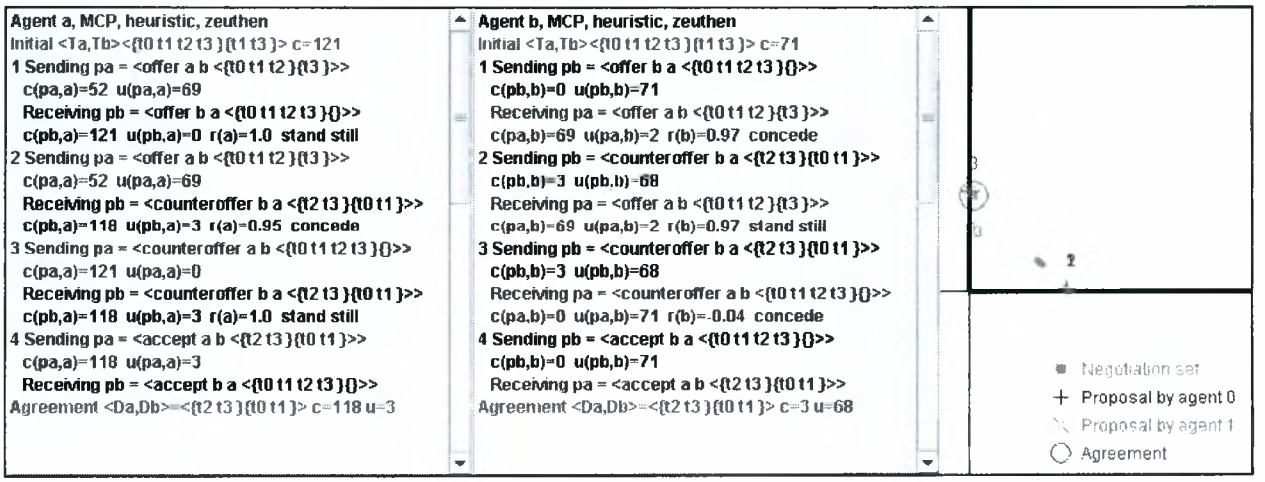
Στο σενάριο 4 το αποτέλεσμα της διαπραγμάτευσης με την ΕΜ είναι ελαφρώς διαφορετικό από αυτό των μεθόδων ΕΜΑ και ΨΜΑ.

Σενάριο 4, ΜΠΣ & Zeuthen, ΕΜ.



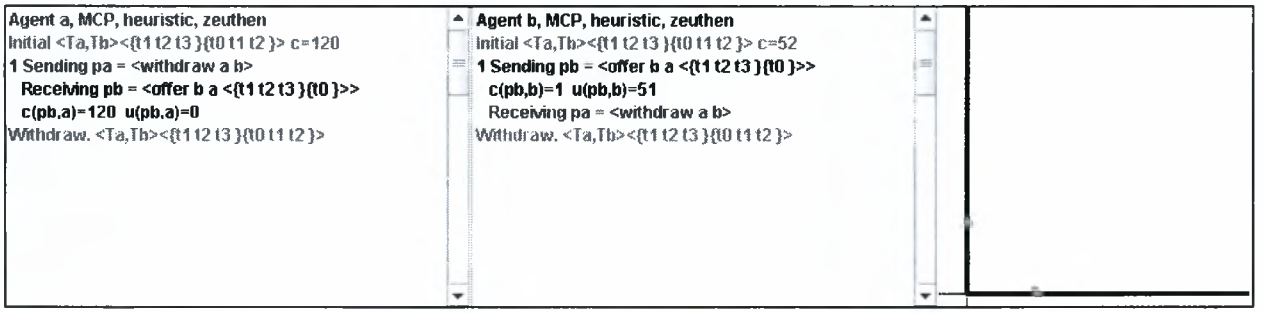
Στο σενάριο 3 ο αριθμός των γύρων είναι μικρότερος από ότι με χρήση των ΕΜΑ, ΨΜΑ και ΠΜΨ, αλλά το αποτέλεσμα είναι πολύ διαφορετικό. Πιο συγκεκριμένα ο πράκτορας a συμβιβάστηκε πολύ περισσότερο απ' όσο έπρεπε.

Σενάριο 3, ΜΠΣ & Zeuthen, ΕΜ.



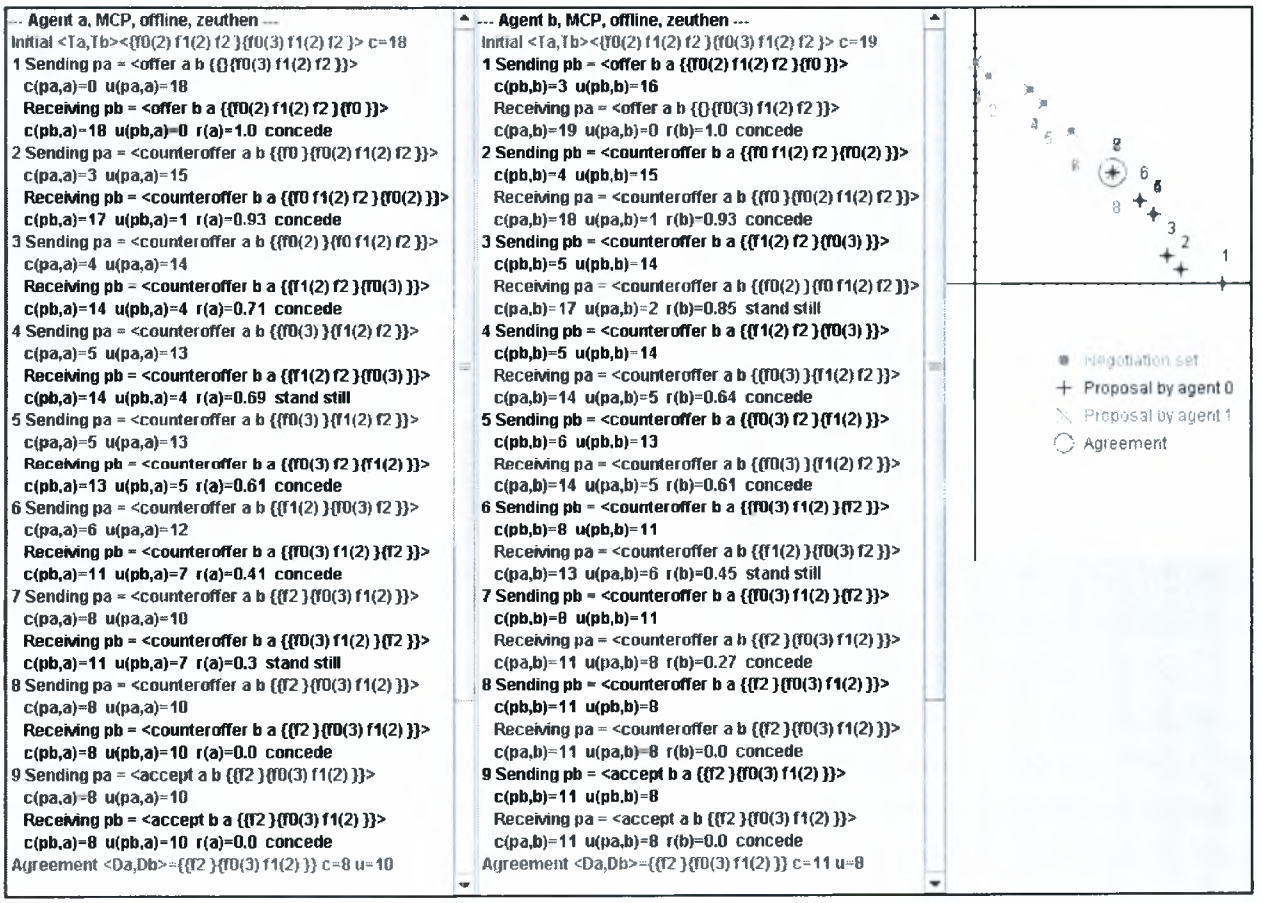
Στο σενάριο 3.1 με την ΕΜ προκύπτει σύγκρουση, ενώ υπάρχει στο σύνολο διαπραγμάτευσης πρόταση ικανοποιητική και για τους δύο πράκτορες. Με τις ΕΜΑ, ΨΜΑ και ΠΜΨ προέκυψε όπως είδαμε τελική λύση.

Σενάριο 3.1, ΜΠΣ & Zeuthen, ΕΜ.

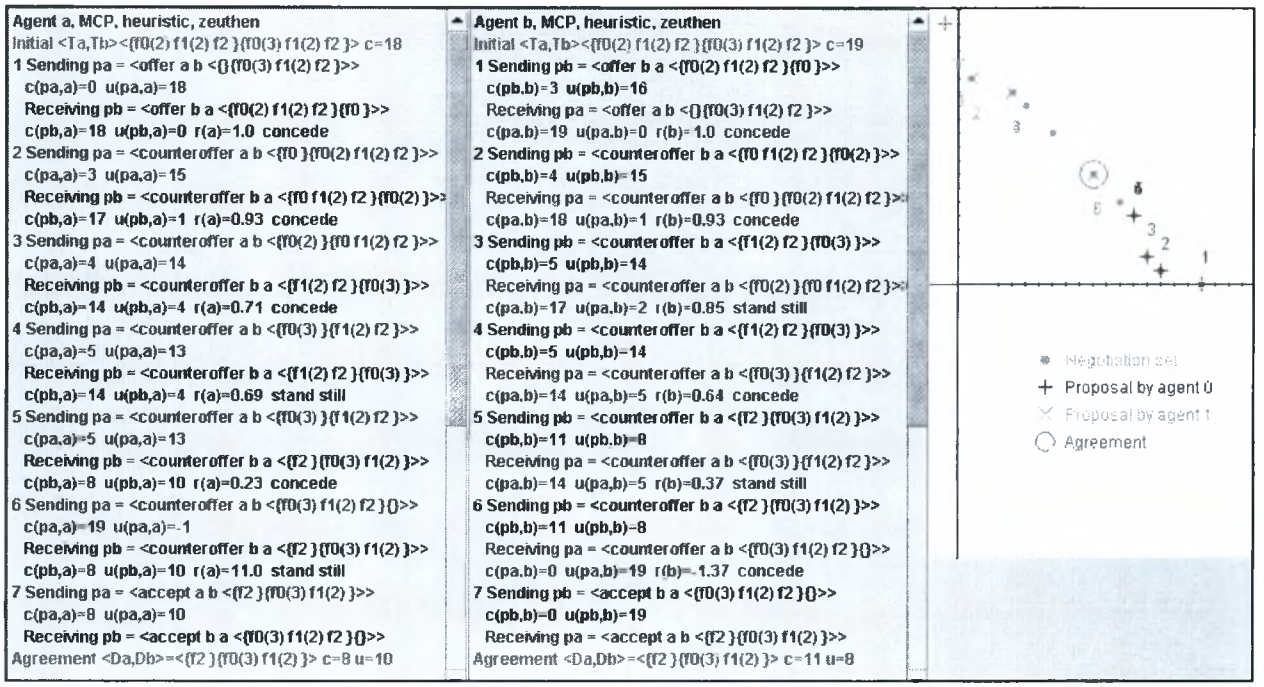


Τέλος στο σενάριο 5 για σύνθετες εργασίες, το τελικό αποτέλεσμα συμπίπτει με το βέλτιστο από πλευράς χρησιμότητας για τους δυο πράκτορες. Όμως, οι ενδιάμεσες προτάσεις δεν ανήκουν καν στο σύνολο διαπραγμάτευσης, γιατί δεν είναι pareto βέλτιστες. Από το σχήμα επαληθεύεται ότι τα **x** και + δεν ταυτίζονται με τις κουκίδες.

Σενάριο 5, ΜΠΣ & Zeuthen, ΕΜΑ & ΨΜΑ.



Σενάριο 5, ΜΠΣ & Zeuthen, ΕΜ.



Συνδυασμός μεθόδων.

Το ΜΠΣ δεν υποχρεώνει να χρησιμοποιούν και οι δυο πράκτορες την ίδια μέθοδο. Στο σημείο αυτό αξίζει τον κόπο να πειραματιστούμε με συνδυασμούς μεθόδων και να δούμε τι αποτέλεσμα προκύπτει.

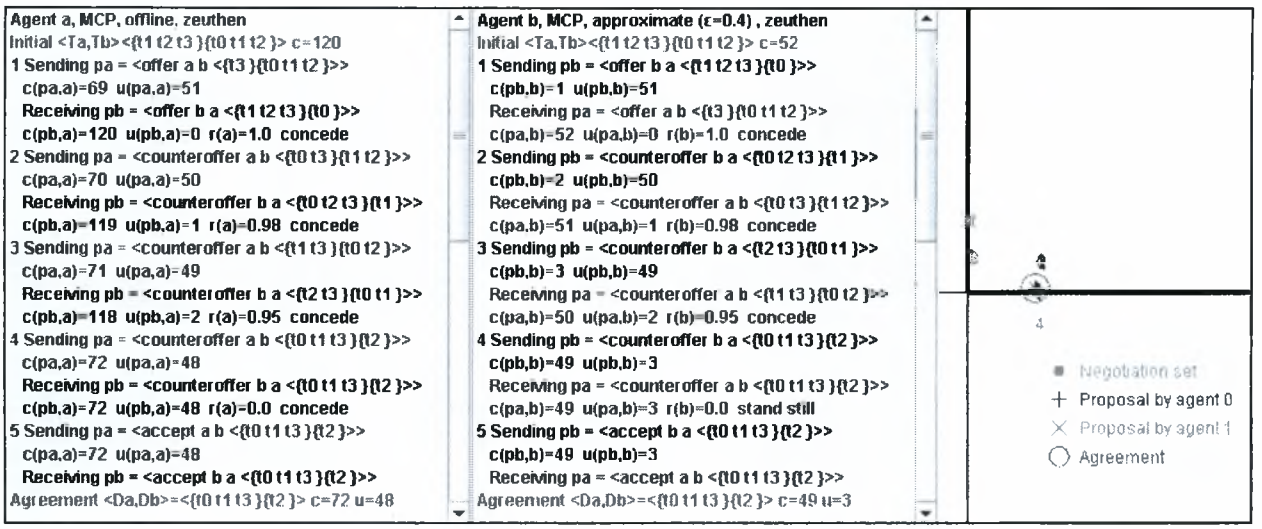
Καταρχήν, οι ΕΜΑ και ΨΜΑ υπολογίζουν όλες τις προτάσεις του συνόλου διαπραγμάτευσης. Επομένως, είτε και οι δυο πράκτορες χρησιμοποιούν την ΕΜΑ, είτε και οι δυο την ΨΜΑ, είτε ο ένας την ΕΜΑ και ο άλλος την ΨΜΑ, τόσο τα ενδιάμεσα όσο και το τελικό αποτέλεσμα θα είναι τα ίδια.

Στη συνέχεια θα εξετάσουμε την περίπτωση στην οποία ο ένας πράκτορας χρησιμοποιεί την ΕΜΑ και ο άλλος την ΠΜΨ ή την ΕΜ.

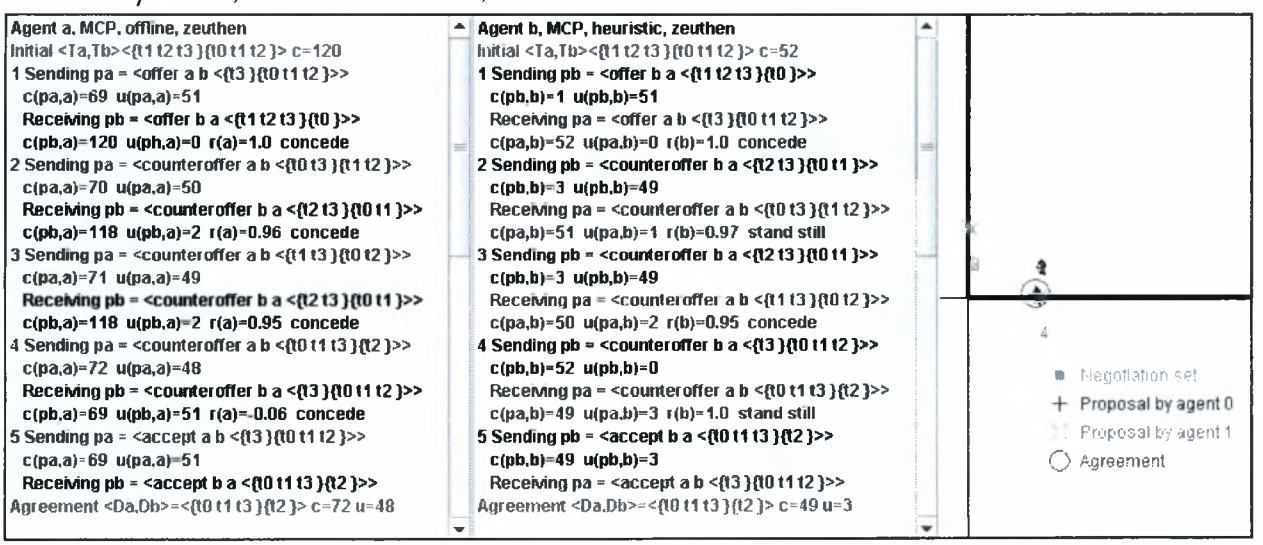
Στο σενάριο 3.1 όταν και οι δυο πράκτορες χρησιμοποιούσαν την ΠΜΨ προέκυπτε αποτέλεσμα αρκετά διαφορετικό από το βέλτιστο. Επίσης, όταν και οι δυο χρησιμοποιούσαν την ΕΜ προέκυπτε σύγκρουση. Βλέπουμε ότι όταν ο ένας πράκτορας χρησιμοποιεί μέθοδο ακρίβειας και ο άλλος προσεγγιστική, το αποτέλεσμα της διαπραγμάτευσης είναι καλύτερο από ότι αν και οι δυο χρησιμοποιούσαν προσεγγιστική. Μάλιστα στο συγκεκριμένο σενάριο το αποτέλεσμα είναι βέλτιστο.

Πρέπει, όμως, να κάνουμε το σχόλιο ότι στις παραπάνω περιπτώσεις, μπορεί ο πράκτορας b με προσεγγιστική μέθοδο να επεδίωκε πολυωνυμική πολυπλοκότητα, αλλά για να παραλάβει την πρώτη πρόταση του αντιπάλου πρέπει να περιμένει τον a να εκτελέσει τον offline υπολογισμό εκθετικής πολυπλοκότητας. Έτσι, ακόμη και αν ο b κερδίζει χρόνο για να ασχοληθεί με κάτι άλλο, συνολικά το σύστημα των δυο πρακτόρων δεν κερδίζει ιδιαίτερα από πλευράς ταχύτητας και απλότητας. Συνεπώς αν οι δυο πράκτορες χρησιμοποιούν διαφορετική μέθοδο διαφορετικής πολυπλοκότητας, το σύστημα συνολικά θα έχει πολυπλοκότητα τη χειρότερη από τις δυο.

Σενάριο 3.1, ΜΠΣ & Zeuthen, ΕΜΑ - ΠΜΨ.

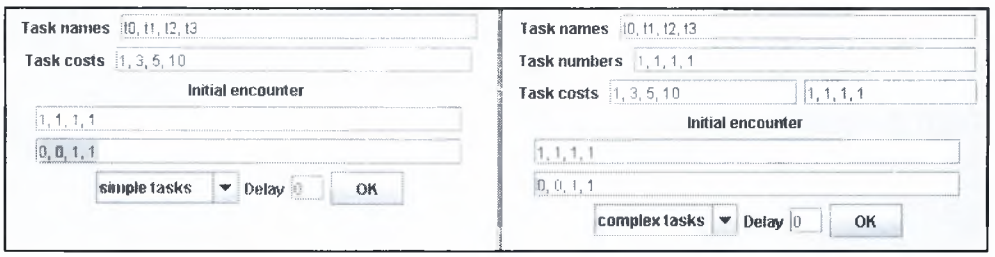


Σενάριο 3.1, ΜΠΣ & Zeuthen, ΕΜΑ - ΕΜ.



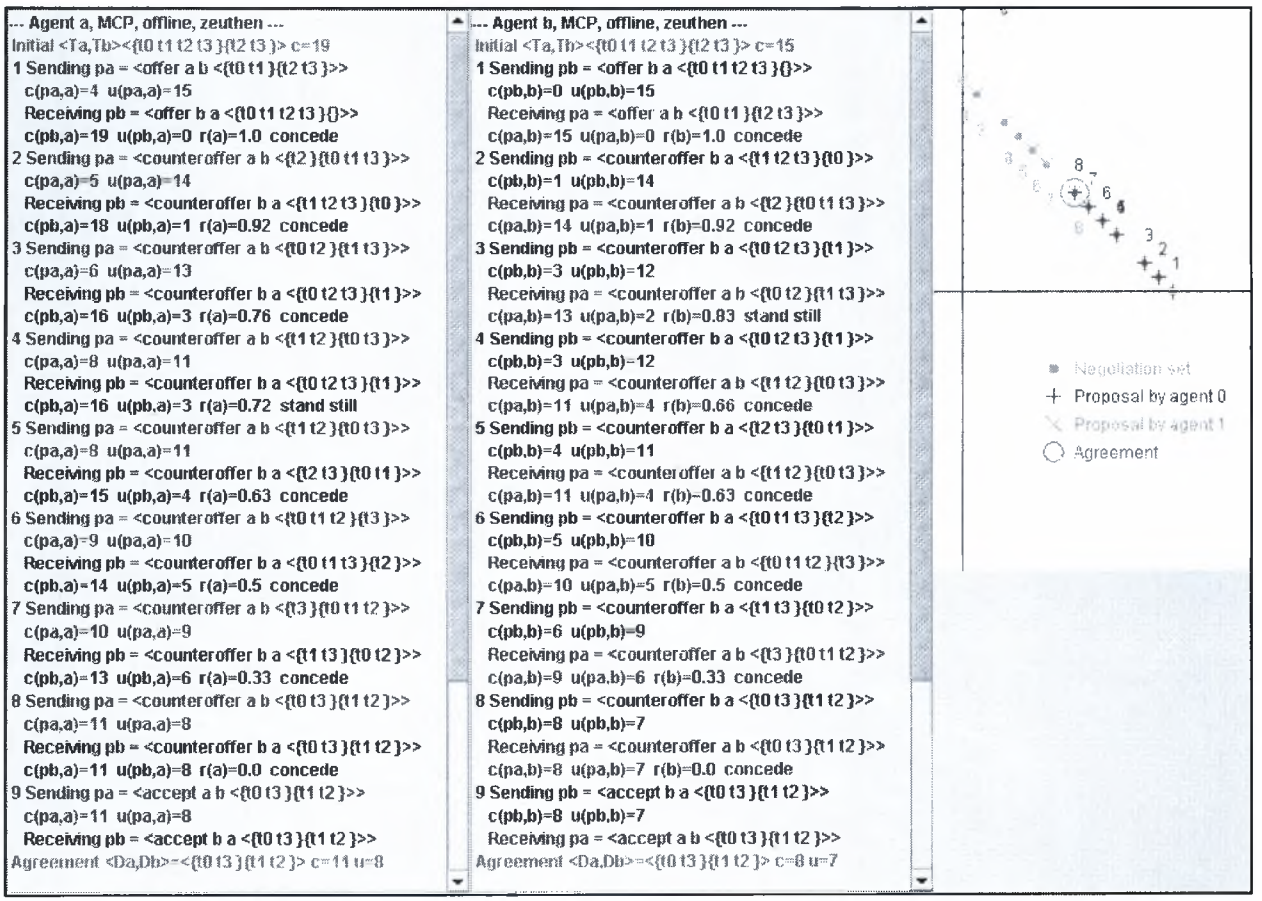
Αντιμετώπιση απλών εργασιών ως σύνθετες.

Στο σημείο αυτό αξίζει να πειραματιστούμε να αντιμετωπίσουμε τις απλές εργασίες των προηγούμενων σεναρίων ως σύνθετες. Αυτό θα εφαρμόσουμε στο σενάριο 1, δηλαδή αντί για τις $\{t_0, t_1, t_2, t_3\}$ θεωρούμε ότι έχουμε τις $\{t_0(1), t_1(1), t_2(1), t_3(1)\}$. Έτσι προκύπτει το σενάριο 1.3 για το οποίο η εισαγωγή δεδομένων στο σύστημα γίνεται όπως στην εικόνα.

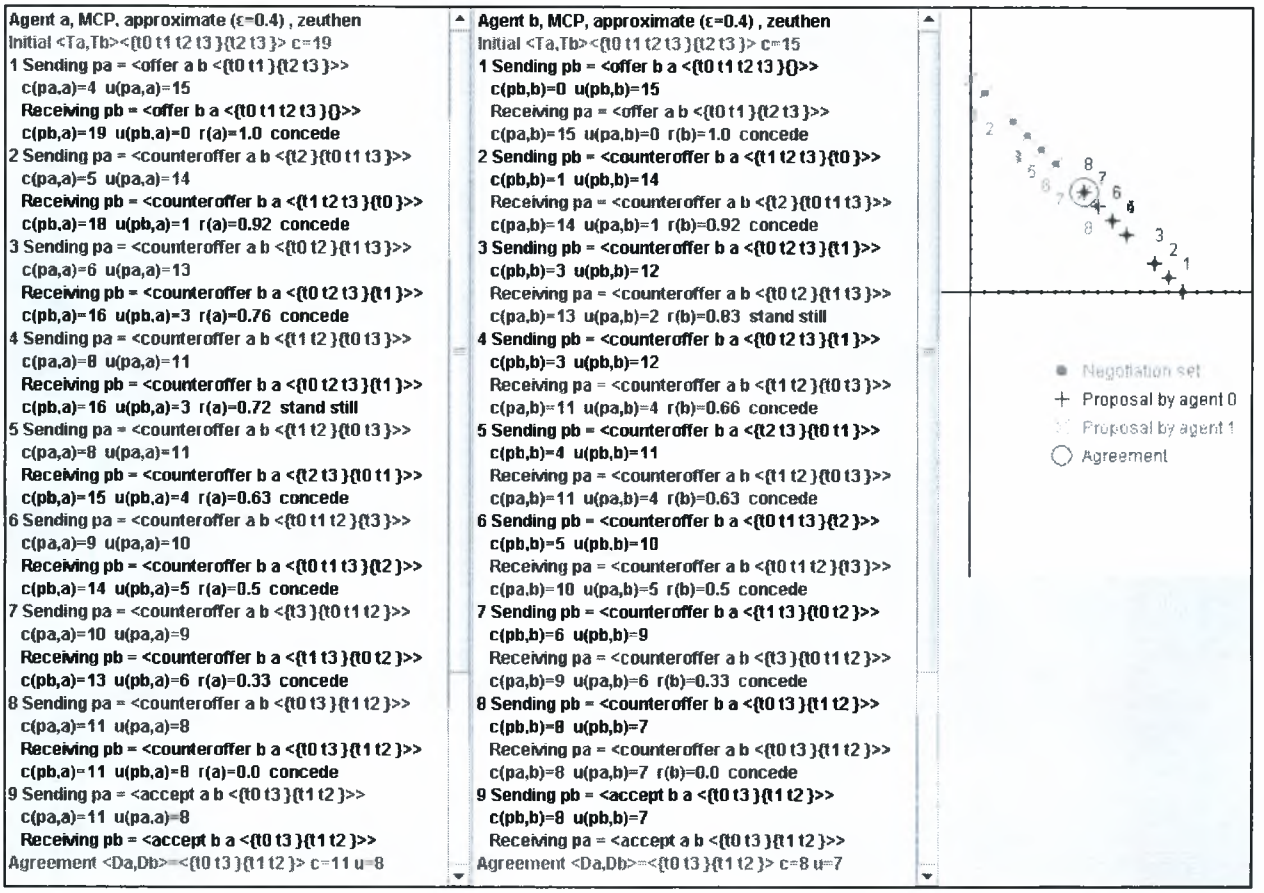


Εσωτερικά στο σύστημα η πληροφορία αναπαρίσταται ως: taskCost = {{1,3,5,10}{0,0,0,0}}, taskNumber = {1,1,1,1}, ie = {{1,1,1,1},{0,0,1,1}}, costI = {1,3,5,10} και dependency = {-1,-1,-1,-1}. Εφαρμόζοντας τις τέσσερις μεθόδους διαπιστώνουμε ότι τα αποτελέσματα τόσο κάθε βήματος όσο και τα τελικά είναι ακριβώς ίδια με αυτά που πήραμε αντιμετωπίζοντας τις εργασίες ως απλές.

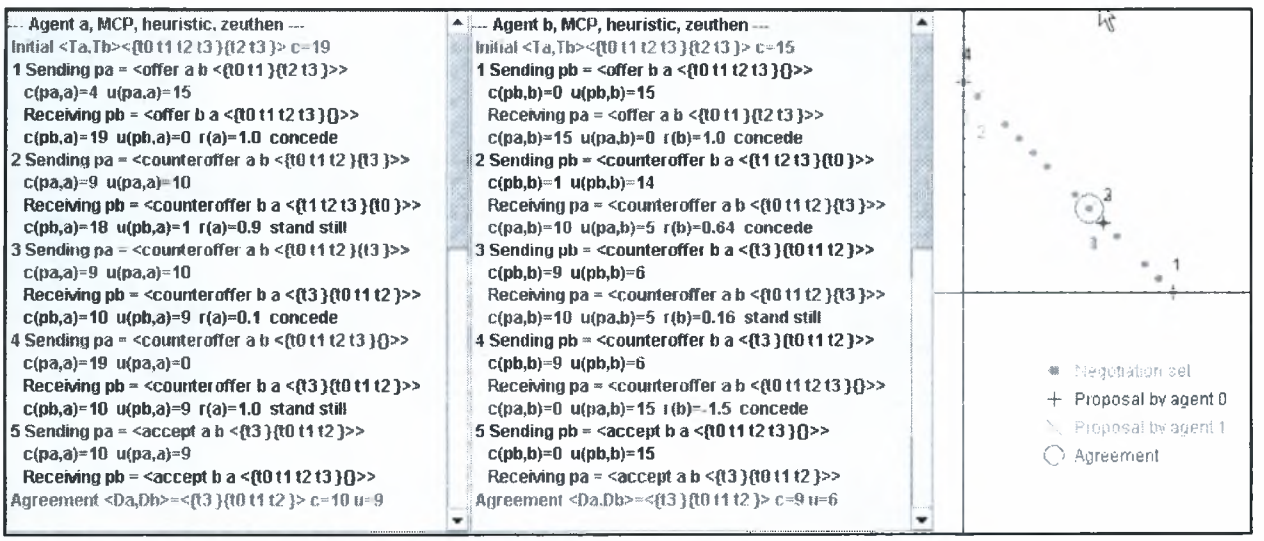
Σενάριο 1.3, ΜΠΣ & Zeuthen, EMA.



Σενάριο 1.3, ΜΠΣ & Zeuthen, ΠΜΨ.



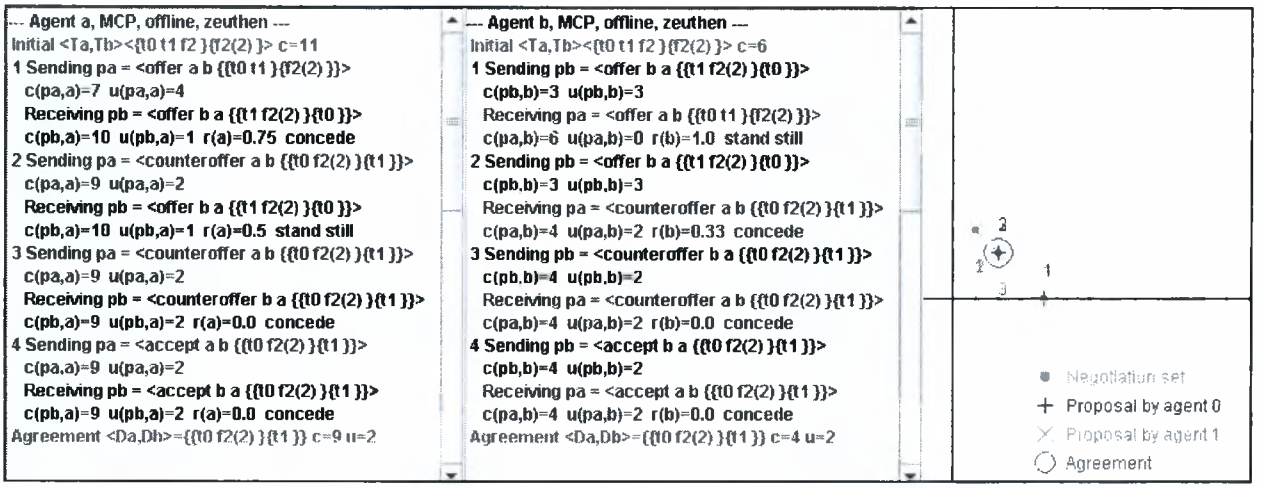
Σενάριο 1.3, ΜΠΣ & Zeuthen, ΕΜ.



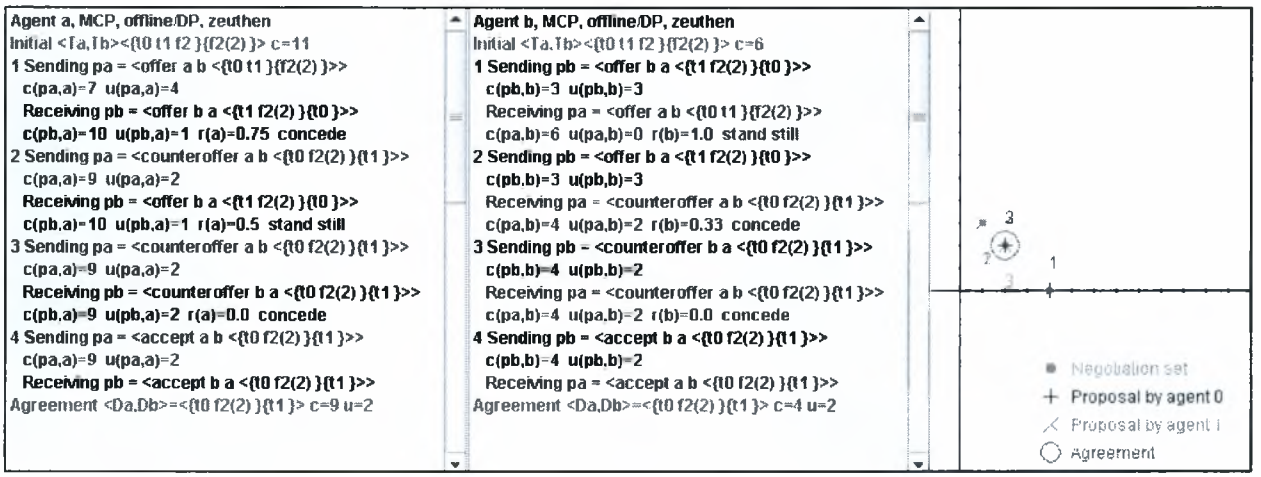
Συνδυασμός απλών και σύνθετων εργασιών.

Μέχρι τώρα εφαρμόσαμε τις τέσσερις μεθόδους σε ΤΠΕ είτε μόνο απλών είτε μόνο σύνθετων εργασιών. Στη συνέχεια θα ασχοληθούμε με ΤΠΕ συνδυασμού απλών και σύνθετων εργασιών. Αυτό είναι εφικτό, εφόσον όπως είδαμε μπορούμε να αντιμετωπίσουμε τις απλές εργασίες ως σύνθετες.

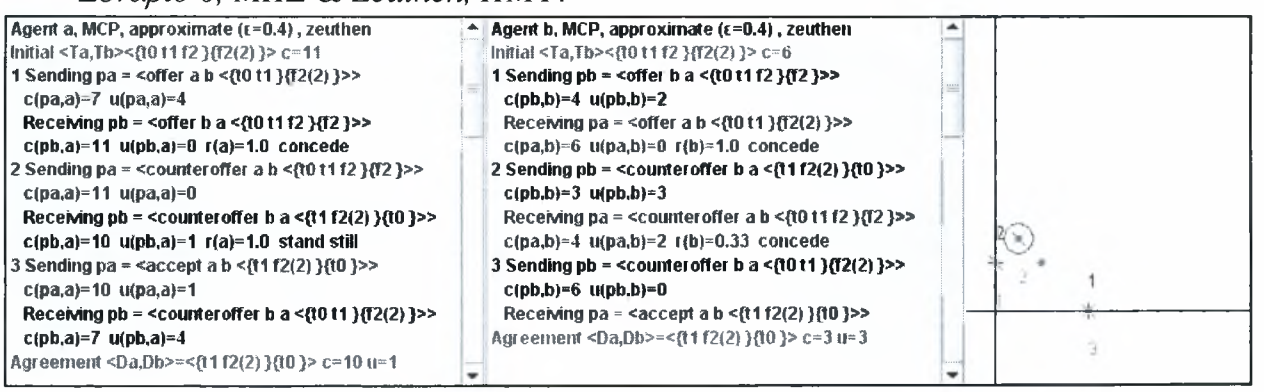
Σενάριο 6, ΜΠΣ & Zeuthen, ΕΜΑ.



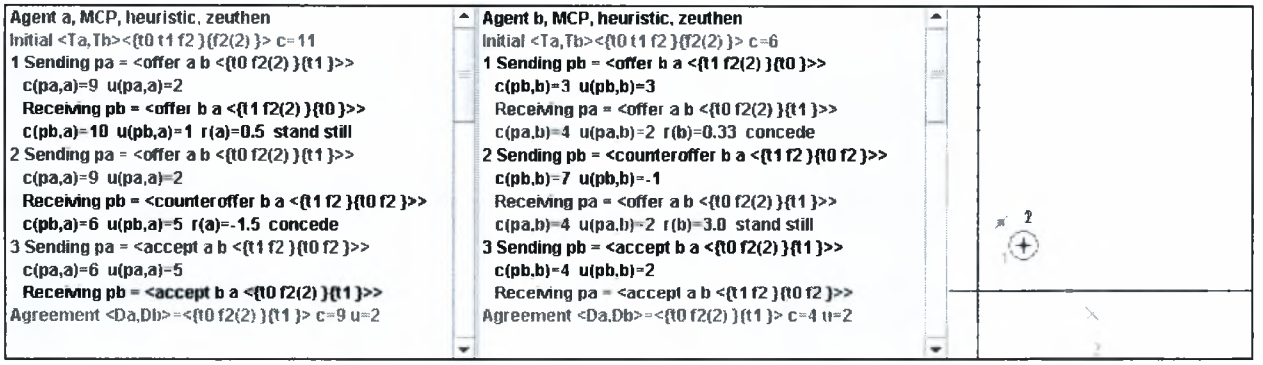
Σενάριο 6, ΜΠΣ & Zeuthen, ΨΜΑ.



Σενάριο 6, ΜΠΣ & Zeuthen, ΠΜΨ.



Σενάριο 6, ΜΠΣ & Zeuthen, ΕΜ.



Πειραματική αποτίμηση των μεθόδων.

Συμπερασματικά, τόσο για απλές, τόσο για σύνθετες εργασίες, όσο και για συνδυασμό των δύο, ισχύουν τα παρακάτω συμπεράσματα.

Οι ΕΜΑ και ΨΜΑ δίνουν τα ίδια ακριβώς αποτελέσματα και τερματίζουν με τον ίδιο αριθμό βημάτων. Είναι παρόμοιες ως προς τη μέθοδο ανάκτησης επόμενης καλύτερης πρότασης από τη λίστα και διαφέρουν ριζικά ως προς τη διαδικασία offline υπολογισμού. Η ΕΜΑ έχει πολυπλοκότητα χρόνου $O(2^{2N})$ και χώρου $O(2^{N})$. Η ΨΜΑ έχει πολυπλοκότητα χρόνου και χώρου O(NW).

Η ΠΜΨ απαιτεί κατά κανόνα μικρότερο offline υπολογισμό για υποψήφιες προτάσεις και η ΕΜ δεν χρειάζεται καθόλου offline υπολογισμό, εκτός από μια ποεραιτική ταξινόμηση. Και οι δυο τερματίζουν συνήθως σε λιγότερα βήματα από τις ΕΜΑ και ΨΜΑ. Παράγουν όμως στη γενική περίπτωση υποβέλτιστα αποτελέσματα. Ειδικά για τις σύνθετες εργασίες, τόσο οι ενδιάμεσες προτάσεις, όσο και η τελική συμφωνία μπορεί να μην ανήκουν καν στο σύνολο διαπραγμάτευσης, γιατί δεν είναι pareto βέλτιστες. Η ΕΜ είναι πιο γρήγορη από την ΠΜΨ, όμως είναι κατά κανόνα πιο αναξιόπιστη. Μπορεί να αποτύχει ακόμη και αν υπάρχει στο σύνολο διαπραγμάτευσης λύση που συμφέρει και τους δυο πράκτορες.

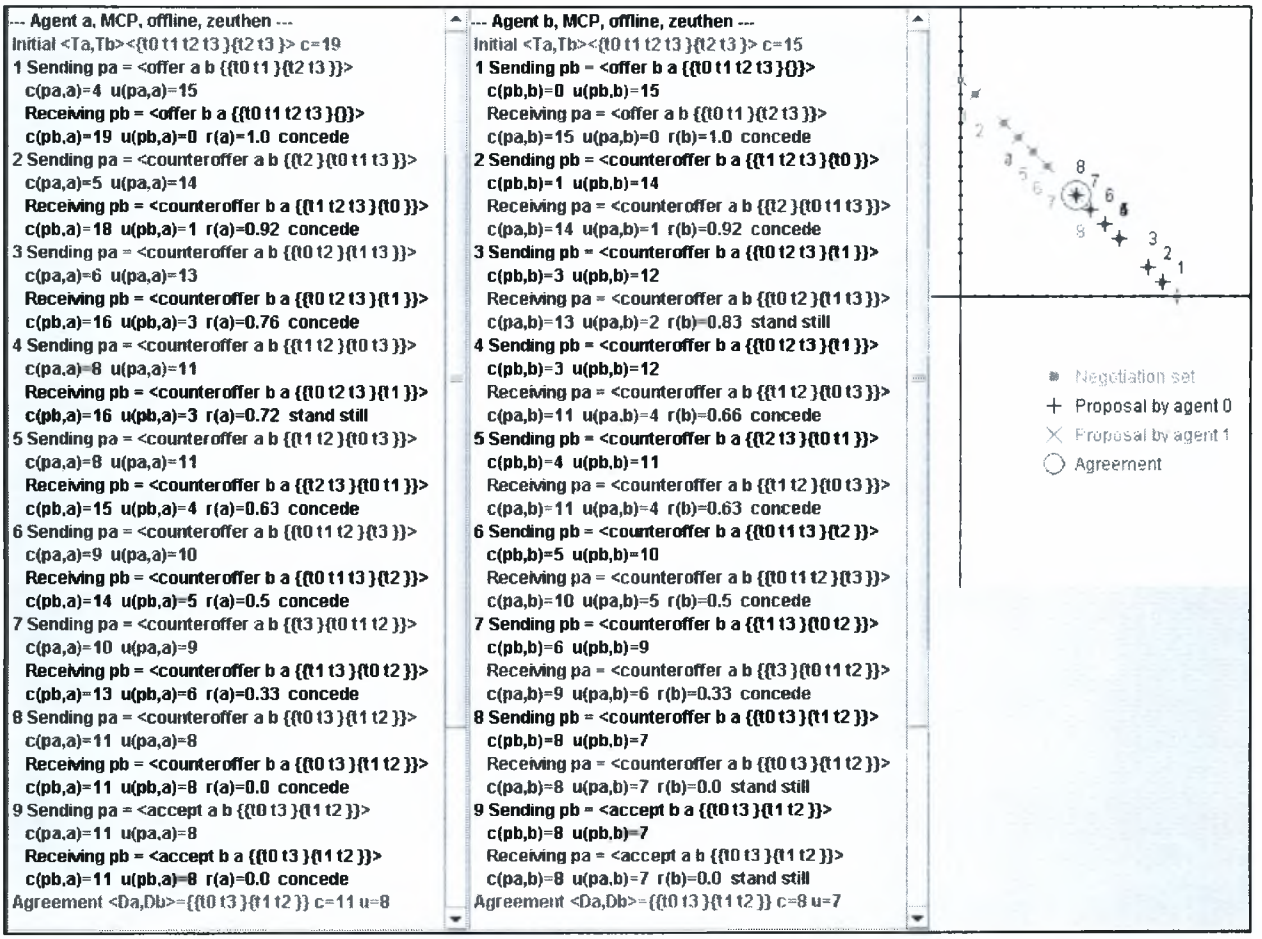
Κάθε μέθοδος έχει τα δικά της χαρακτηριστικά. Κάθε πράκτορας μπορεί να επιλέξει ποια μέθοδο θα χρησιμοποιήσει, ανάλογα με τη φύση του προβλήματος, δηλαδή τις τιμές N και W, και με βάση τις προτιμήσεις και προτεραιότητές του, δηλαδή αν προτιμά ταχύτητα ή ακρίβεια λύσης. Τέλος, δεν είναι απαραίτητο να χρησιμοποιήσουν και οι δυο πράκτορες την ίδια μέθοδο, αν και η αντίθετη περίπτωση δεν συμφέρει ιδιαίτερα το σύστημα των πρακτόρων.

4.2. Σύγκριση πολιτικών υποχώρησης.

Με χρήση της πολιτικής 1, δηλαδή της στρατηγικής Zeuthen, σε κάθε βήμα υποχωρεί ο πράκτορας με τη μικρότερη τιμή υπολογιζόμενου ρίσκου και σε περίπτωση ισότητας και οι δυο.

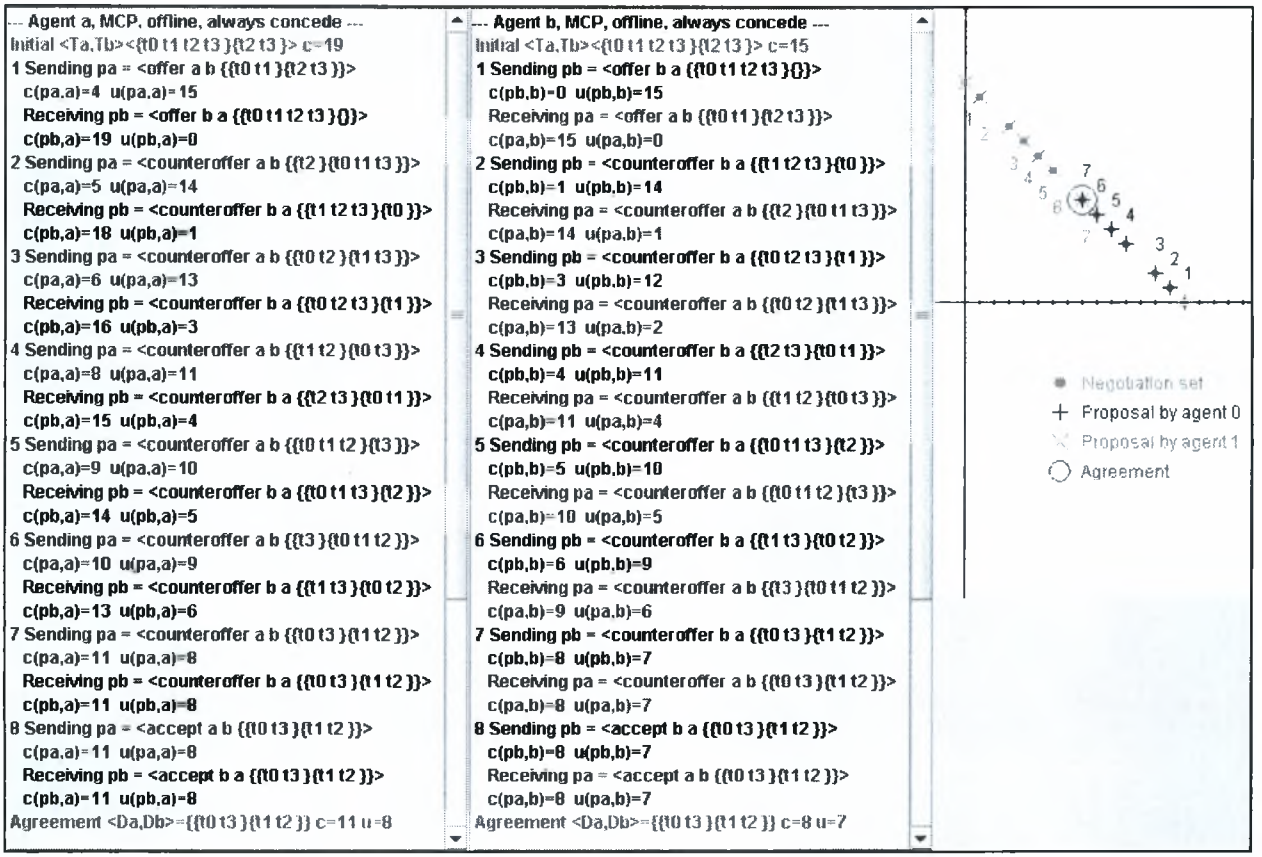
Στο σενάριο 1.1, κατά το γύρο 1 έχουμε στον a: $r_a = (u_a(\delta_a) - u_a(\delta_b)) / u_a(\delta_a) = 1$ και $r_b = (u_b(\delta_b) - u_b(\delta_a)) / u_b(\delta_b) = 1$. Επειδή $r_a = r_b$ στον επόμενο γύρο υποχωρούν και οι δυο. Κατά τον ίδιο τρόπο στο γύρο 4 υποχωρεί μόνον ο a και στο γύρο 5 μόνον ο b.

Σενάριο 1.1, ΜΠΣ & Zeuthen, ΕΜΑ.

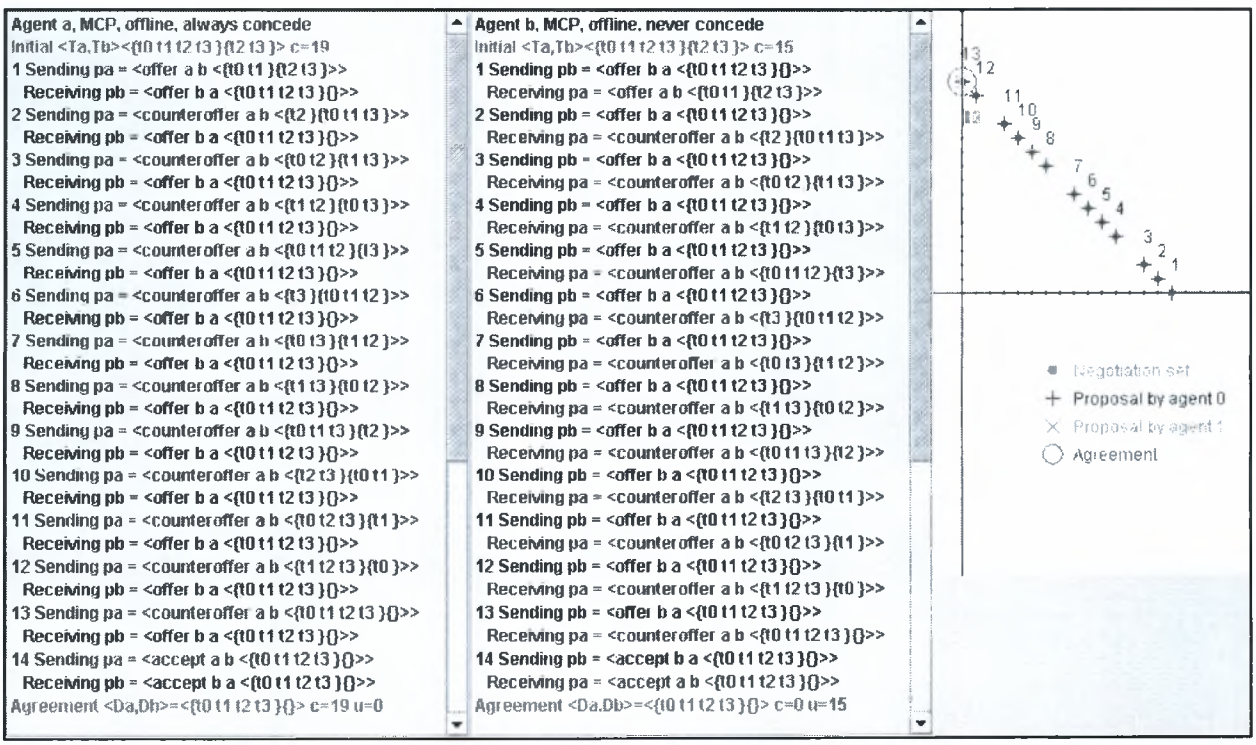


Στη συνέχεια θα χρησιμοποιήσουμε την πολιτική 2, σύμφωνα με την οποία υποχωρεί κανείς πάντα σε κάθε γύρο. Ανάλογα το σενάριο, τα τελικά αποτελέσματα μπορεί να είναι ίδια ή διαφορετικά με και χωρίς τη στρατηγική Zeuthen. Σίγουρα, αν σε κάθε βήμα υποχωρούν και οι δυο, τα βήματα θα είναι λιγότερα απ' ότι με τη Zeuthen με την οποία μπορεί να υποχωρεί μόνο ένας (εκτός αν έχουν ίδιο ρίσκο).

Σενάριο 1.1, ΜΠΣ & συνεχής υποχώρηση, ΕΜΑ.



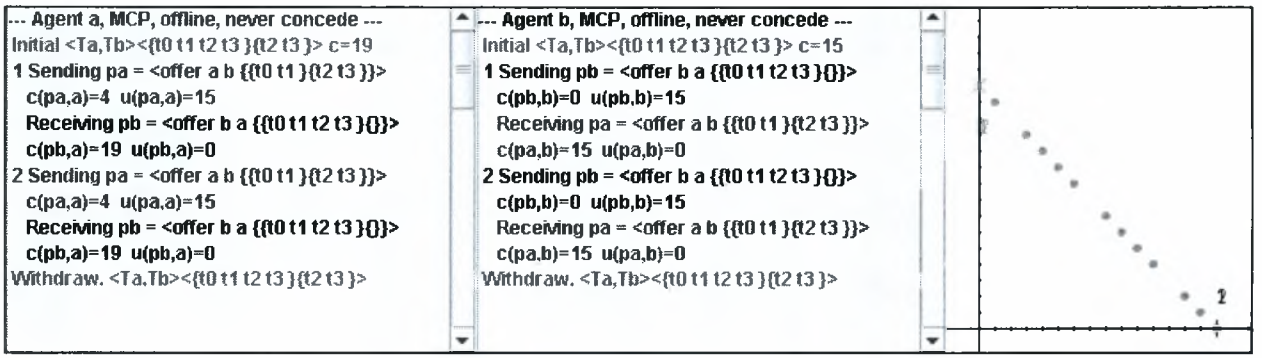
Σενάριο 1.1, ΜΠΣ & συνεχής - μηδενική υποχώρηση, ΕΜΑ.



Το βασικό μειονέκτημα της πολιτικής 2 έναντι της Zeuthen είναι, όπως προαναφέρθηκε ότι μπορεί ο αντίπαλος να την εκμεταλλευτεί και να επιλέξει μηδενική υποχώρηση σε κάθε βήμα (πολιτική 3), ώστε τελικά να επιβάλει τη δική του πρώτη προτίμηση με μέγιστη χρησιμότητα για τον ίδιο, όπως φαίνεται στο σενάριο 1.1. Μάλιστα, ο a είναι ο μεγάλος χαμένος, γιατί, χωρίς να καταφέρει καθόλου να βελτιώσει την κατάστασή του, επιβαρύνεται και με τον offline υπολογισμό και τους 14 γύρους της διαπραγμάτευσης.

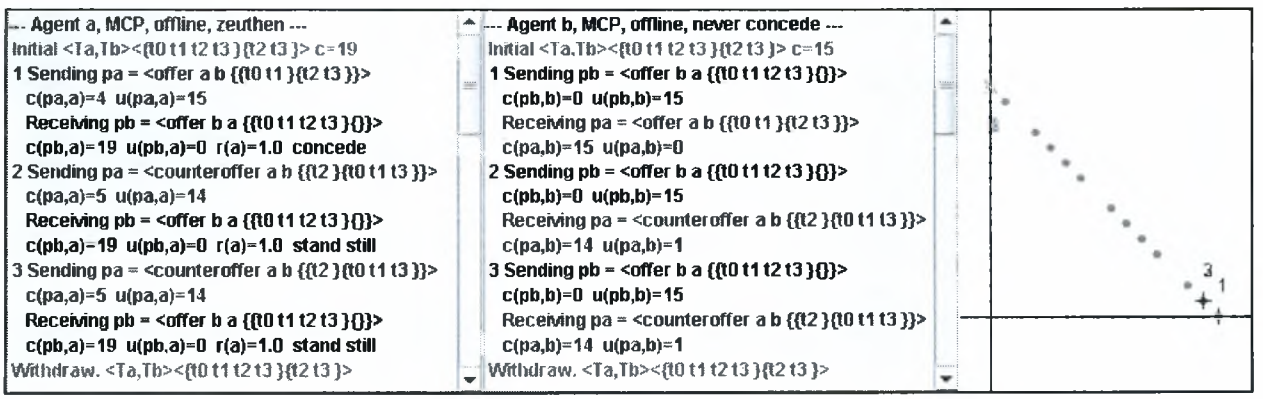
Αναφορικά με την πολιτική 3, είναι προφανές ότι δεν έχει νόημα να τη χρησιμοποιούν και οι δυο πράκτορες ταυτόχρονα, γιατί θα καταλήξουν οπωσδήποτε σε σύγκρουση.

Σενάριο 1.1, ΜΠΣ & Zeuthen - μηδενική υποχώρηση, ΕΜΑ.



Είδαμε προηγουμένως ότι μπορεί κανείς να εφαρμόσει την πολιτική 3 για να εκμεταλλευτεί έναν αντίπαλο που γνωρίζει ότι χρησιμοποιεί την πολιτική 2. Αν όμως ο αντίπαλος χρησιμοποιεί τη στρατηγική Zeuthen, δεν μπορεί να τον ξεγελάει συνεχώς ο πράκτορας που χρησιμοποιεί την πολιτική 3, όπως φαίνεται στο επόμενο σενάριο. Αν μάλιστα γνωρίζει ο πράκτορας ότι ο αντίπαλος χρησιμοποιεί Zeuthen, κρίνει ότι δεν τον συμφέρει να χρησιμοποιήσει την πολιτική 3, γιατί θα ρισκάρει σύγκρουση.

Σενάριο 1.1, ΜΠΣ & Zeuthen - μηδενική υποχώρηση, ΕΜΑ.



Αναφορικά με την πολιτική 4 τυχαίας υποχώρησης, υπάρχει η δυνατότητα αλλαγής του συντελεστή σύμφωνα με τον οποίο ένας πράκτορας θα υποχωρήσει. Για παράδειγμα, συντελεστής 60% σημαίνει ότι ο πράκτορας έχει πιθανότητα 60% να υποχωρήσει στον επόμενο γύρο και 40% να ξανακάνει την ίδια πρόταση. Κάθε πράκτορας μπορεί να έχει ξεχωριστό, διαφορετικό συντελεστή. Υπενθυμίζεται ότι αν σε κάποιο γύρο δεν υποχωρήσει κανείς, τότε η διαπραγμάτευση τερματίζει.

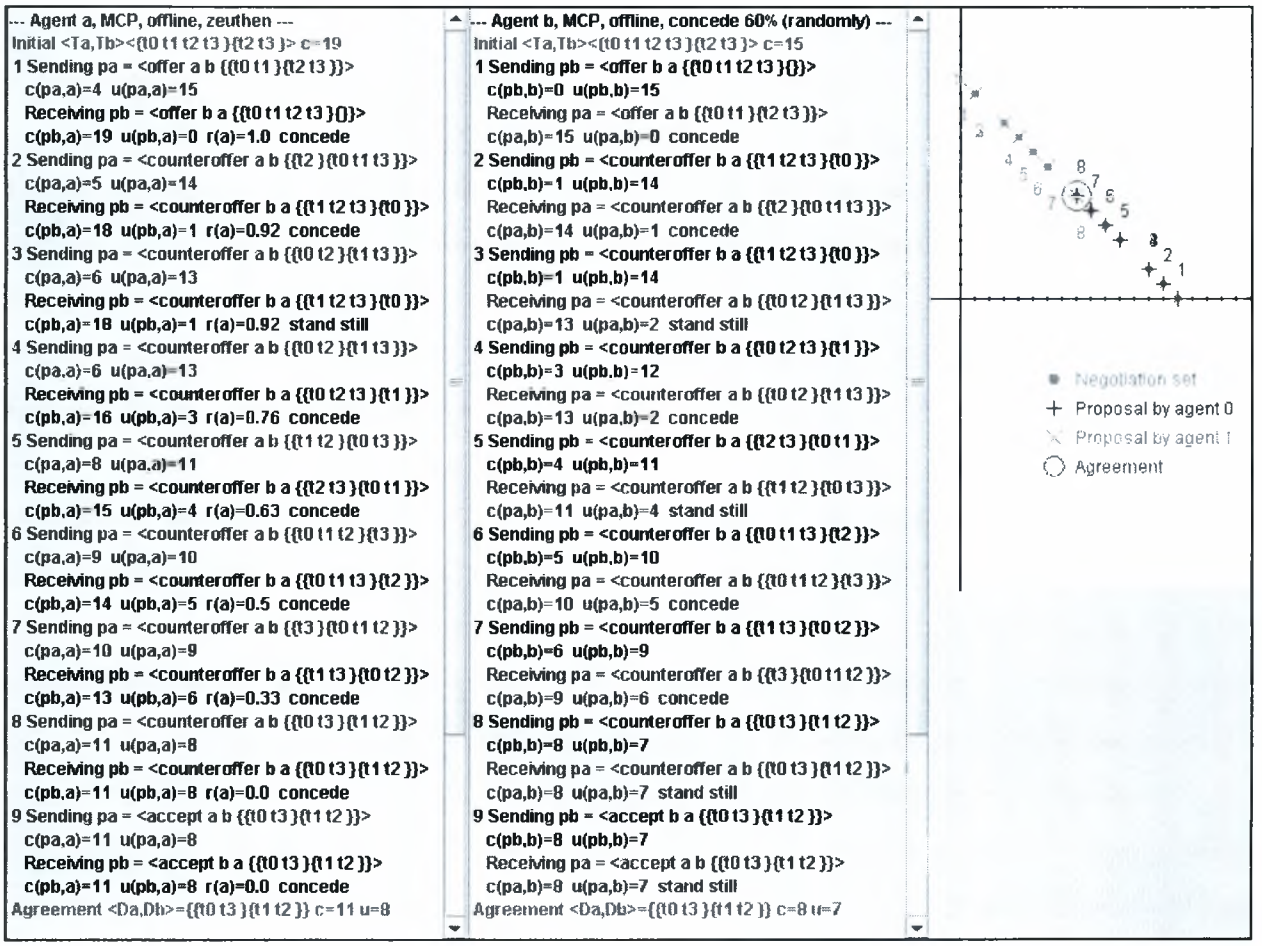
Υπάρχουν πολλοί διαφορετικοί συνδυασμοί που μπορούμε να δοκιμάσουμε για τους δυο πράκτορες χρησιμοποιώντας τον ίδιο ή διαφορετικούς συντελεστές, π.χ. 50-50, 100-60, 30-90, κοκ. Ακόμη και αν καταλήξουμε σε ένα συνδυασμό, όμως, επειδή υπάρχει ο παράγοντας τυχαιότητας το αποτέλεσμα θα είναι διαφορετικό σε κάθε τρέξιμο, είναι δηλαδή μη ντετερμινιστικό.

Πραγματοποιήθηκαν δοκιμές με χρήση της πολιτικής 4 με τους συντελεστές 50-50 και 60-60 για το γνωστό σενάριο 1.1. Υπενθυμίζεται ότι το σενάριο τερματίζει επιτυχώς με την πολιτική 1 σε 9 βήματα και με την πολιτική 2 σε 8. Το τελικό αποτέλεσμα ήταν {{t0,t3}{t1,t2}} με χρησιμότητα 8 και 7. Ακολουθούν τα αποτελέσματα:

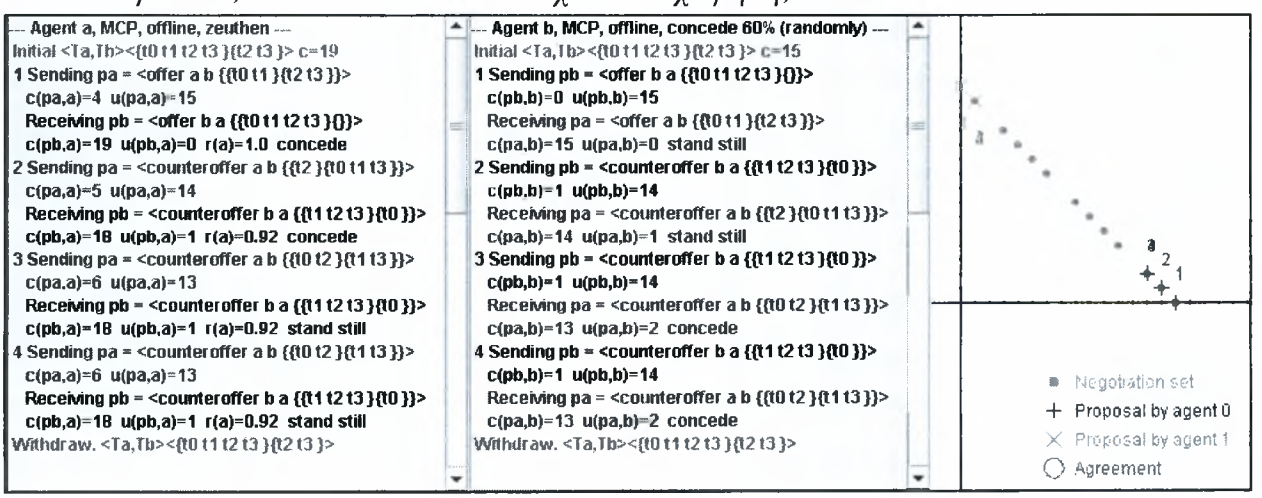
Για τους συντελεστές 50-50, από τα 10 τρεξίματα, τα 8 κατέληξαν σε ασυμφωνία. Τα υπόλοιπα είναι: <{t1,t3}{t0,t2}> με u0=6, u1=9 σε 10 βήματα και <{t0,t1,t3}{t2}> με u0=5, u1=10 σε 10 βήματα. Για τους συντελεστές 60-60, από τα 10 τρεξίματα, τα 6 κατέληξαν σε ασυμφωνία. Τα υπόλοιπα είναι: <{t0,t3}{t1,t2}> με u0=8, u1=7 σε 12 βήματα, <{t1,t3}{t0,t2}> με u0=6, u1=9 σε 11 βήματα, <{t1,t2}{t0,t3}> με u0=11, u1=4 σε 12 βήματα και <{t1,t2}{t0,t3}> με u0=11, u1=4 σε 13 βήματα.

Σημειώνεται ότι δεν συμφέρει ένα πράκτορα να χρησιμοποιήσει αυτή την πολιτική αν γνωρίζει ότι ο αντίπαλος δεν υποχωρεί ποτέ. Κι αυτό γιατί σε κάποιο βήμα δεν θα υποχωρήσει κανείς και θα εγκαταλειφθεί η διαπραγμάτευση. Ακόμη και αν στην ακραία περίπτωση αυτό δεν συμβεί ο πράκτορας θα οδηγηθεί να συμφωνήσει να εκτελέσει τις εργασίες που του δόθηκαν κατά την αρχική αλληλεπίδραση. Από την άλλη, ο πράκτορας δεν έχει τίποτα να χάσει αν την χρησιμοποιήσει όταν ο αντίπαλος υποχωρεί πάντα. Το τελικό αποτέλεσμα θα τον συμφέρει σίγουρα. Αναφορικά με συνδυασμό της πολιτικής 4 με την στρατηγική Zeuthen, πάλι το αποτέλεσμα δεν είναι προβλέψιμο και πάντα το ίδιο. Ακολουθούν δυο διαφορετικά τρεξίματα με τις ίδιες παραμέτρους.

Σενάριο 1.1, ΜΠΣ & Zeuthen - τυχαία υποχώρηση, ΕΜΑ.



Σενάριο 1.1, ΜΠΣ & Zeuthen - τυχαία υποχώρηση, ΕΜΑ.



Στο σημείο αυτό αξίζει να κάνουμε κάποιο σχόλιο σχετικά με την ελάχιστη ικανή υποχώρηση που απαιτείται κατά τη διάρκεια της διαπραγμάτευσης. Κάθε πράκτορας επιλέγει σε κάθε γύρο να υποχωρεί κατά το ελάχιστο δυνατό, ώστε να κερδίσει όσο γίνεται περισσότερο, όχι όμως σε βάρος του αντίπαλου. Ειδικά στη στρατηγική Zeuthen, μπορούμε σε κάθε γύρο να σχηματίζουμε τη νέα πρόταση με την ελάχιστη υποχώρηση και υπολογίζουμε τη νέα τιμή ρίσκου. Αν δεν έχει αλλάξει η ισορροπία ρίσκου, τότε γίνεται νέα ελάχιστη υποχώρηση, κοκ.

Εφαρμόζουμε το σενάριο 1.2. Στην πρώτη εικόνα οι πράκτορες δεν εφαρμόζουν την ελάχιστη ικανή υποχώρηση. Έτσι στο βήμα 6 με βάση την τιμή ρίσκου πρέπει να υποχωρήσει ο b, και στο βήμα 7 με βάση την τιμή ρίσκου πρέπει να υποχωρήσει πάλι ο b. Στη δεύτερη εικόνα βλέπουμε το ίδιο σενάριο με ελάχιστη ικανή υποχώρηση. Ο πράκτορας προτιμά να υποχωρήσει δυο φορές στο βήμα 6 κι έτσι συνολικά γλιτώνουμε ένα βήμα.

Σενάριο 1.2, ΜΠΣ & Zeuthen με ελάχιστη ικανή υποχώρηση, ΕΜΑ

Initial <Ta,Tb><{t1 t2 t3}{t0 t1 t3}> c=28	Initial <Ta,Tb><{t1 t2 t3}{t0 t1 t3}> c=24
1 Sending offer a b <{t2}{t0 t1 t3}> c=5 u=23	1 Sending offer b a <{t1 t2 t3}{t0}> c=1 u=23
Evaluating offer b a <{t1 t2 t3}{t0}> c=28 u=0 r=1.0	Evaluating offer a b <{t2}{t0 t1 t3}> c=24 u=0 r=1.0
2 Sending counteroffer a b <{t0 t2}{t1 t3}> c=6 u=22	2 Sending counteroffer b a <{t0 t2 t3}{t1}> c=3 u=21
Evaluating counteroffer b a <{t0 t2 t3}{t1}> c=26 u=2 r=0.9	Evaluating counteroffer a b <{t0 t2}{t1 t3}> c=23 u=1 r=0.95
3 Sending counteroffer a b <{t1 t2}{t0 t3}> c=8 u=20	3 Sending counteroffer b a <{t0 t2 t3}{t1}> c=3 u=21
Evaluating counteroffer b a <{t0 t2 t3}{t1}> c=26 u=2 r=0.9	Evaluating counteroffer a b <{t1 t2}{t0 t3}> c=21 u=3 r=0.85
4 Sending counteroffer a b <{t1 t2}{t0 t3}> c=8 u=20	4 Sending counteroffer b a <{t2 t3}{t0 t1}> c=4 u=20
Evaluating counteroffer b a <{t2 t3}{t0 t1}> c=25 u=3 r=0.85	Evaluating counteroffer a b <{t1 t2}{t0 t3}> c=21 u=3 r=0.85
5 Sending counteroffer a b <{t0 t1 t2}{t3}> c=9 u=19	5 Sending counteroffer b a <{t0 t1 t3}{t2}> c=5 u=19
Evaluating counteroffer b a <{t0 t1 t3}{t2}> c=24 u=4 r=0.78	Evaluating counteroffer a b <{t0 t1 t2}{t3}> c=20 u=4 r=0.78
6 Sending counteroffer a b <{t3}{t0 t1 t2}> c=20 u=8	6 Sending counteroffer b a <{t1 t3}{t0 t2}> c=6 u=18
Evaluating counteroffer b a <{t1 t3}{t0 t2}> c=23 u=5 r=0.37	Evaluating counteroffer a b <{t3}{t0 t1 t2}> c=9 u=15 r=0.16
7 Sending counteroffer a b <{t3}{t0 t1 t2}> c=20 u=8	7 Sending counteroffer b a <{t0 t3}{t1 t2}> c=8 u=16
Evaluating counteroffer b a <{t0 t3}{t1 t2}> c=21 u=7 r=0.12	Evaluating counteroffer a b <{t3}{t0 t1 t2}> c=9 u=15 r=0.06
8 Sending counteroffer a b <{t3}{t0 t1 t2}> c=20 u=8	8 Sending counteroffer b a <{t3}{t0 t1 t2}> c=9 u=15
Evaluating counteroffer b a <{t3}{t0 t1 t2}> c=20 u=8 r=0.0	Evaluating counteroffer a b <{t3}{t0 t1 t2}> c=9 u=15 r=0.0
9 Sending accept a b <{t3}{t0 t1 t2}> c=20 u=8	9 Sending accept b a <{t3}{t0 t1 t2}> c=9 u=15
Evaluating accept b a <{t3}{t0 t1 t2}> c=20 u=8 r=0.0	Evaluating accept a b <{t3}{t0 t1 t2}> c=9 u=15 r=0.0
Agreement <Da,Db>= <{t3}{t0 t1 t2}> c=20 u=8	Agreement <Da,Db>= <{t3}{t0 t1 t2}> c=9 u=15

Σενάριο 1.2, ΜΠΣ & Zeuthen χωρίς ελάχιστη ικανή υποχώρηση, ΕΜΑ

Agent a, method 1, zeuthen	Agent b, method 1, zeuthen
Initial <Ta,Tb><{t1 t2 t3}{t0 t1 t3}> c=28	Initial <Ta,Tb><{t1 t2 t3}{t0 t1 t3}> c=24
1 Sending offer a b <{t2}{t0 t1 t3}> c=5 u=23	1 Sending offer b a <{t1 t2 t3}{t0}> c=1 u=23
Evaluating offer b a <{t1 t2 t3}{t0}> c=28 u=0 r=1.0	Evaluating offer a b <{t2}{t0 t1 t3}> c=24 u=0 r=1.0
2 Sending counteroffer a b <{t0 t2}{t1 t3}> c=6 u=22	2 Sending counteroffer b a <{t0 t2 t3}{t1}> c=3 u=21
Evaluating counteroffer b a <{t0 t2 t3}{t1}> c=26 u=2 r=0.9	Evaluating counteroffer a b <{t0 t2}{t1 t3}> c=23 u=1 r=0.95
3 Sending counteroffer a b <{t1 t2}{t0 t3}> c=8 u=20	3 Sending counteroffer b a <{t0 t2 t3}{t1}> c=3 u=21
Evaluating counteroffer b a <{t0 t2 t3}{t1}> c=26 u=2 r=0.9	Evaluating counteroffer a b <{t1 t2}{t0 t3}> c=21 u=3 r=0.85
4 Sending counteroffer a b <{t1 t2}{t0 t3}> c=8 u=20	4 Sending counteroffer b a <{t2 t3}{t0 t1}> c=4 u=20
Evaluating counteroffer b a <{t2 t3}{t0 t1}> c=25 u=3 r=0.85	Evaluating counteroffer a b <{t1 t2}{t0 t3}> c=21 u=3 r=0.85
5 Sending counteroffer a b <{t0 t1 t2}{t3}> c=9 u=19	5 Sending counteroffer b a <{t0 t1 t3}{t2}> c=5 u=19
Evaluating counteroffer b a <{t0 t1 t3}{t2}> c=24 u=4 r=0.78	Evaluating counteroffer a b <{t0 t1 t2}{t3}> c=20 u=4 r=0.78
6 Sending counteroffer a b <{t3}{t0 t1 t2}> c=20 u=8	6 Sending counteroffer b a <{t1 t3}{t0 t2}> c=6 u=18
Evaluating counteroffer b a <{t1 t3}{t0 t2}> c=23 u=5 r=0.37	Evaluating counteroffer a b <{t3}{t0 t1 t2}> c=9 u=15 r=0.16
7 Sending counteroffer a b <{t3}{t0 t1 t2}> c=20 u=8	7 Sending counteroffer b a <{t3}{t0 t1 t2}> c=9 u=15
Evaluating counteroffer b a <{t3}{t0 t1 t2}> c=20 u=8 r=0.0	Evaluating counteroffer a b <{t3}{t0 t1 t2}> c=9 u=15 r=0.0
8 Sending accept a b <{t3}{t0 t1 t2}> c=20 u=8	8 Sending accept b a <{t3}{t0 t1 t2}> c=9 u=15
Evaluating accept b a <{t3}{t0 t1 t2}> c=20 u=8 r=0.0	Evaluating accept a b <{t3}{t0 t1 t2}> c=9 u=15 r=0.0
Agreement <Da,Db>= <{t3}{t0 t1 t2}> c=20 u=8	Agreement <Da,Db>= <{t3}{t0 t1 t2}> c=9 u=15

Πειραματική αποτίμηση των πολιτικών.

Συγκρίνοντας τις τέσσερις πολιτικές καταλήγουμε στα εξής συμπεράσματα: Η πολιτική συνεχούς υποχώρησης μπορεί να γίνει αφορμή εκμετάλλευσης από τον αντίπαλο. Η πολιτική μηδενικής υποχώρησης προκαλεί εύκολα σύγκρουση στη διαδικασία της διαπραγμάτευσης. Η πολιτική τυχαίας υποχώρησης δίνει μη ντετερμινιστικά αποτελέσματα, δηλαδή κάθε τρέξιμο με το ίδιο σενάριο μπορεί να λειτουργήσει εντελώς διαφορετικά και μπορεί επίσης εύκολα να προκληθεί σύγκρουση. Συνεπώς, η στρατηγική Zeuthen είναι η πλέον αξιόπιστη και αποτελεσματική πολιτική υποχώρησης, χωρίς να προκαλεί υπερβολική υπολογιστική επιβάρυνση στο σύστημα.

Ανακεφαλαίωση.

Στο κεφάλαιο αυτό πειραματιστήκαμε με σενάρια ΤΠΕ που αφορούν απλές και σύνθετες εργασίες. Πραγματοποιήθηκε σύγκριση των χρησιμοποιούμενων μεθόδων εύρεσης καλύτερης επόμενης πρότασης και πολιτικών υποχώρησης. Έτσι, επαληθεύτηκαν και πρακτικά τα θεωρητικά συμπεράσματα του προηγούμενου κεφαλαίου.

Στο επόμενο κεφάλαιο γίνεται απόπειρα να συγκρίνουμε το ΜΠΣ με τις συγκεκριμένες μεθόδους και πολιτικές με ένα εναλλακτικό πρωτόκολλο.

5. Πρωτόκολλο ενός βήματος.

Στη βιβλιογραφία περιγράφεται το *Πρωτόκολλο ενός Βήματος*[30] (one-step protocol - ΠΕΒ), σύμφωνα με το οποίο και οι δυο πράκτορες κάνουν ταυτόχρονα μια πρόταση. Θα συμφωνήσουν εξ ορισμού με την πρόταση που μεγιστοποιεί το γινόμενο των χρησιμοτήτων των πρακτόρων και σε περίπτωση ισότητας θα επιλεγεί αυθαίρετα η μία πρόταση.

Αλγόριθμος 5.1: ΠΕΒ

Προαιρετικός offline υπολογισμός με χρήση της κατάλληλης μεθόδου *Παραγωγή πιο συμφέρουσας για τον πράκτορα πρότασης που μεγιστοποιεί το γινόμενο χρησιμοτήτων με χρήση της κατάλληλης μεθόδου* Αποστολή στον αντίπαλο Παραλαβή πρότασης αντιπάλου Αποτίμηση προτάσεων

Κάθε πράκτορας που χρησιμοποιεί το συγκεκριμένο πρωτόκολλο ακολουθεί την επόμενη στρατηγική: Από όλες τις πιθανές προτάσεις βρίσκει ποιες μεγιστοποιούν το γινόμενο των χρησιμοτήτων των δυο πρακτόρων. Από αυτές, επιλέγει την πιο συμφέρουσα για τον ίδιο, δηλαδή με τη μέγιστη χρησιμότητα, και την προτείνει στον αντίπαλο.

Αναφορικά με την εύρεση των υποψήφιων προτάσεων, μπορεί να χρησιμοποιηθεί κάποια από τις τέσσερις μεθόδους που αναλύθηκαν στο ΜΠΣ. Σημειώνεται ότι με τις ΕΜΑ και ΨΜΑ μπορούν να εντοπιστούν όλες οι προτάσεις του συνόλου διαπραγμάτευσης και να πραγματοποιηθεί – offline ή μη – υπολογισμός του γινομένου των χρησιμοτήτων για καθεμιά από αυτές. Επίσης, θα μπορούσε να προκύψει μια προσεγγιστική λύση με την ΠΜΨ ή την ΕΜ, υπολογίζοντας μια-μια τις πιθανές προτάσεις και το γινόμενο χρησιμοτήτων.

Επειδή το ΠΕΒ περιλαμβάνει ένα βήμα μόνο, δεν έχει νόημα η χρήση πολιτικών υποχώρησης, όπως έγινε στο ΜΠΣ.

Στη συνέχεια θα εφαρμόσουμε στο σενάριο 3 του προηγούμενου κεφαλαίου το ΠΕΒ σε συνδυασμό με τις τέσσερις μεθόδους.

[30] [Rosenschein & Zlotkin 1994, σ.49]

Σενάριο 3.

Έστω οι εργασίες $\{t_0, t_1, t_2, t_3\}$ με κόστος {1, 2, 49, 69}. Αρχική ανάθεση e = $<\{t_0, t_1, t_2, t_3\}\ \{t_1, t_3\}>$. Ισχύει $c_a(e) = 121$ και $c_b(e) = 71$. Αρχικό σημείο (a) = 121–71 = 50 και αρχικό (b) = 121–121 = 0. Τελικό (a) = 121 και τελικό (b) = 71.

Από τις ΕΜΑ και ΨΜΑ προέκυψε μέσω offline υπολογισμού η λίστα υποσυνόλων ταξινομημένη κατά αύξουσα σειρά κόστους. Για τον πράκτορα a είναι $\{\{t_0,t_2\}$-50, $\{t_1,t_2\}$-51, $\{t_0,t_1,t_2\}$-52, $\{t_3\}$-69, $\{t_0,t_3\}$-70, $\{t_1,t_3\}$-71, $\{t_0,t_1,t_3\}$-72, $\{t_2,t_3\}$-118, $\{t_0,t_2,t_3\}$-119, $\{t_1,t_2,t_3\}$-120, $\{t_0,t_1,t_2,t_3\}$-121$\}$. Στη συνέχεια υπολογίζεται ποιο υποσύνολο της λίστας μεγιστοποιεί το γινόμενο χρησιμοτήτων των δυο πρακτόρων.

Da	$\{t_0,t_2\}$	$\{t_1,t_2\}$	$\{t_0,t_1,t_2\}$	$\{t_3\}$	$\{t_0,t_3\}$	$\{t_1,t_3\}$	$\{t_0,t_1,t_3\}$	$\{t_2,t_3\}$	$\{t_0,t_2,t_3\}$	$\{t_1,t_2,t_3\}$	$\{t_0,t_1,t_2,t_3\}$
ca	50	51	52	69	70	71	72	118	119	120	121
ua	71	70	69	52	51	50	49	3	2	1	0
cb	71	70	69	52	51	50	49	3	2	1	0
ub	0	1	2	19	20	21	22	68	69	70	71
uaxub	0	70	138	988	1020	1050	1078	204	138	70	0

max(uaxub) = 1078 που αντιστοιχεί στην πρόταση $<\{t_0,t_1,t_3\}\{t_2\}>$.

Από την ΠΜΨ προκύπτει η αντίστοιχη λίστα υποσυνόλων ταξινομημένη κατά αύξουσα σειρά κόστους. Για τον πράκτορα a είναι $\{\{t_0,t_2\}$-50, $\{t_3\}$-69, $\{t_2,t_3\}$-118$\}$. Από αυτή υπολογίζεται ποιο υποσύνολο της λίστας μεγιστοποιεί το γινόμενο χρησιμοτήτων των δυο πρακτόρων.

Da	$\{t_0,t_2\}$	$\{t_3\}$	$\{t_2,t_3\}$
ca	50	69	118
ua	71	52	3
cb	71	52	3
ub	0	19	68
uaxub	0	988	204

max(uaxub) = 988 που αντιστοιχεί στην πρόταση $<\{t_3\}\{t_0,t_1,t_2\}>$.

Από την ΕΜ προκύπτει η αντίστοιχη λίστα υποσυνόλων ταξινομημένη κατά αύξουσα σειρά κόστους. Από αυτή υπολογίζεται ποιο υποσύνολο της λίστας μεγιστοποιεί το γινόμενο χρησιμοτήτων των δυο πρακτόρων.

Da	∅	$\{t_0\}$	$\{t_0,t_1\}$	$\{t_0,t_1,t_2\}$	$\{t_0,t_1,t_2,t_3\}$
ca	0 < 50	1 < 50	3 < 50	52	121
ua	απορρίπτεται	απορρίπτεται	απορρίπτεται	69	0
cb				69	0
ub				2	71
uaxub				138	0

max(uaxub) = 138 που αντιστοιχεί στην πρόταση $<\{t_0,t_1,t_2\}\{t_3\}>$.

Σε όλες τις περιπτώσεις η διαπραγμάτευση διαρκεί ένα γύρο. □

Το ΠΕΒ και η προτεινόμενη στρατηγική χαρακτηρίζονται από σταθερότητα. Αν ένας πράκτορας επιλέξει την πρόταση που μεγιστοποιεί το γινόμενο χρησιμοτήτων και τον συμφέρει προσωπικά, τότε ο αντίπαλος δεν μπορεί παρά να κάνει το ίδιο. Επίσης, εξασφαλίζεται συμμετρία. Ακόμη, είναι απλούστερο από το ΜΠΣ γιατί δεν απαιτείται ένα πλήθος γύρων διαπραγμάτευσης. Στην περίπτωση που χρησιμοποιούνται όλες οι προτάσεις του συνόλου διαπραγμάτευσης (δηλαδή με τη χρήση μιας μεθόδου ακρίβειας για τον υπολογισμό επόμενης πρότασης), το ΠΕΒ εγγυάται αποτελεσματικότητα. Η πρόταση που μεγιστοποιεί τη χρησιμότητα των δυο πρακτόρων θα είναι pareto βέλτιστη. Επίσης, η διαπραγμάτευση δεν θα καταλήξει σε σύγκρουση, εφόσον υπάρχει στο σύνολο διαπραγμάτευσης πρόταση συμφέρουσα και για τους δυο. Αυτό μπορεί να συνέβαινε στο ΜΠΣ αν στο τελευταίο βήμα αποδεχόταν και οι δυο την πρόταση του αντιπάλου και επέλεγε κανείς τυχαία αν θα υποχωρούσε ή όχι.

Να σημειωθεί ότι τόσο το ΜΠΣ με Zeuthen στρατηγική, όσο και το ΠΕΒ, ανήκουν στην οικογένεια *Μηχανισμών Μεγιστοποίησης Γινομένου*[31] (ΜΜΓ) που ικανοποιούν τις παρακάτω συνθήκες:

- ✓ το πρωτόκολλο είναι συμμετρικό
- ✓ η στρατηγική χαρακτηρίζεται από σταθερότητα
- ✓ δυο πράκτορες που τα χρησιμοποιούν θα καταλήξουν σε πρόταση που μεγιστοποιεί το γινόμενο των χρησιμοτήτων. Αν υπάρχουν περισσότερες από μια τέτοιες προτάσεις, θα καταλήξουν σε αυτό που μεγιστοποιεί το άθροισμα των χρησιμοτήτων. Αν υπάρχουν περισσότερες από μια τέτοιες προτάσεις, θα καταλήξουν σε κάποιο με τυχαίο τρόπο.

Το σύστημα των δύο πρακτόρων που περιγράγραψαμε στα προηγούμενα κεφάλαια εμπλουτίζεται με την δυνατότητα διαπραγμάτευσης με το ΠΕΒ. Για το νέο σύστημα τα διαγράμματα δραστηριοτήτων περιλαμβάνονται στο Παράρτημα και είναι τα εξής:

- Διάγραμμα Π.1.1: Δραστηριότητα «Διαπραγμάτευση». Αντικαθιστά το Π.1.
- Διαγράμματα Π.2, Π.3, Π.4, Π.5, όπως πριν.
- Διάγραμμα Π.6: Δραστηριότητα «Παραγωγή καλύτερης πρότασης».

Το διαγράμμα κλάσεων (Διάγραμμα Π.7) είναι το ίδιο με πριν.

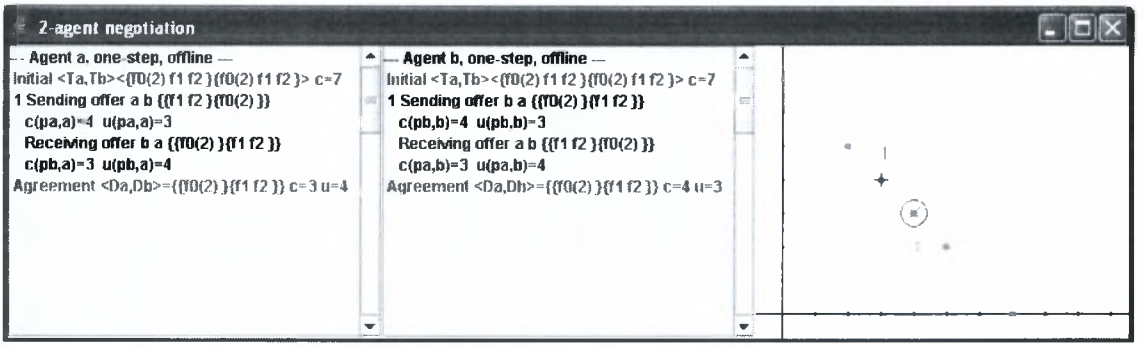
Οι συμβάσεις για τη σχεδίαση και υλοποίηση του συστήματος είναι παρόμοιες με τις προηγούμενες. Η αναπαράσταση της πληροφορίας στο σύστημα πραγματοποιείται με τον ίδιο τρόπο.

---

[31] [Rosenschein & Zlotkin 1994, σ.50]

Αναφορικά με την είσοδο δεδομένων στο σύστημα στο αρχικό παράθυρο καταχώρησης δεδομένων ορίζεται το είδος του πρωτοκόλλου, από τη πτυσσόμενη λίστα. του σχήματος

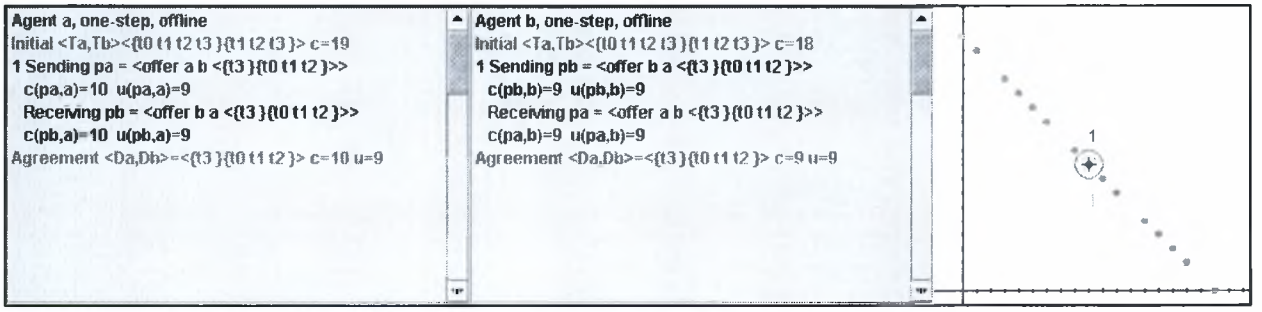
Στο παράθυρο εξόδου πλέον περιλαμβάνεται ένα μόνο βήμα, τόσο στο παράθυρο διαλόγου, όσο και στη γραφική αναπαράσταση.



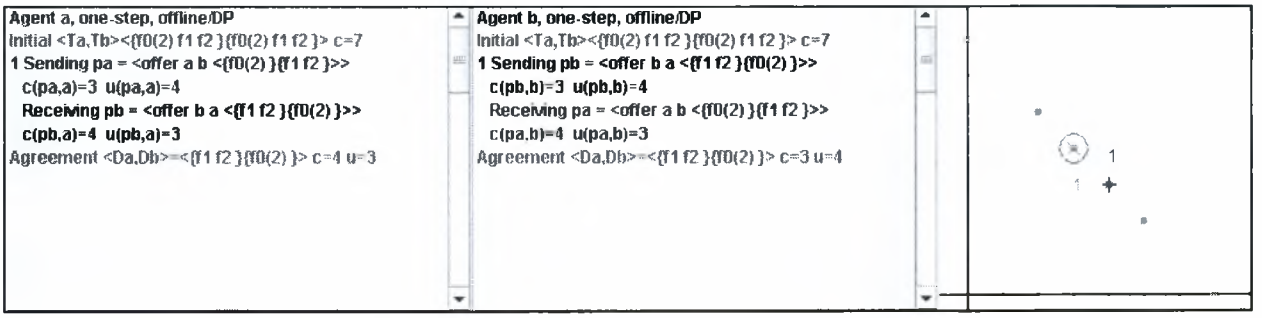
Στη συνέχεια θα μελετήσουμε τη συμπεριφορά του ΠΕΒ σε κάποια από τα σενάρια που περιγράφηκαν στο προηγούμενο κεφάλαιο και θα συγκρίνουμε πειραματικά τα δυο πρωτόκολλα.

Παρατηρούμε ότι με χρήση των ΕΜΑ και ΨΜΑ, το τελικό αποτέλεσμα της διαπραγμάτευσης με ΠΕΒ είναι το ίδιο με αυτό της διαπραγμάτευσης με ΜΠΣ & Zeuthen και μάλιστα προκύπτει σε ένα μόνο γύρο.

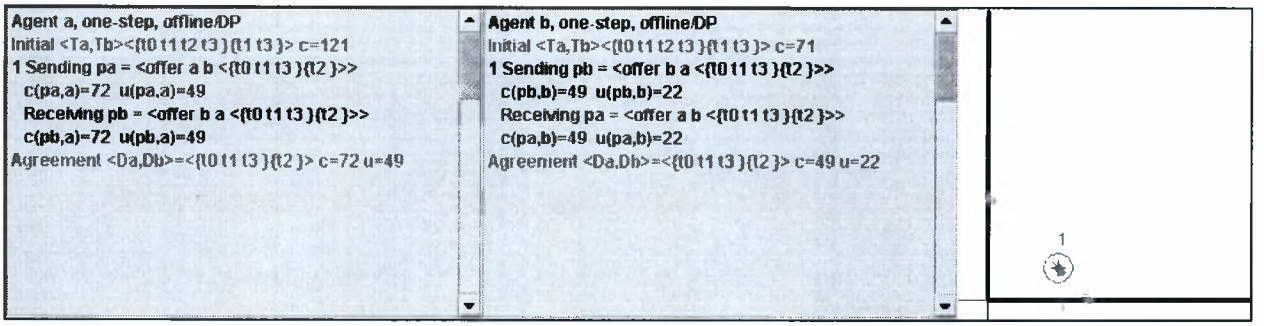
Σενάριο 1, ΠΕΒ, ΕΜΑ.



Σενάριο 2, ΠΕΒ, ΨΜΑ.



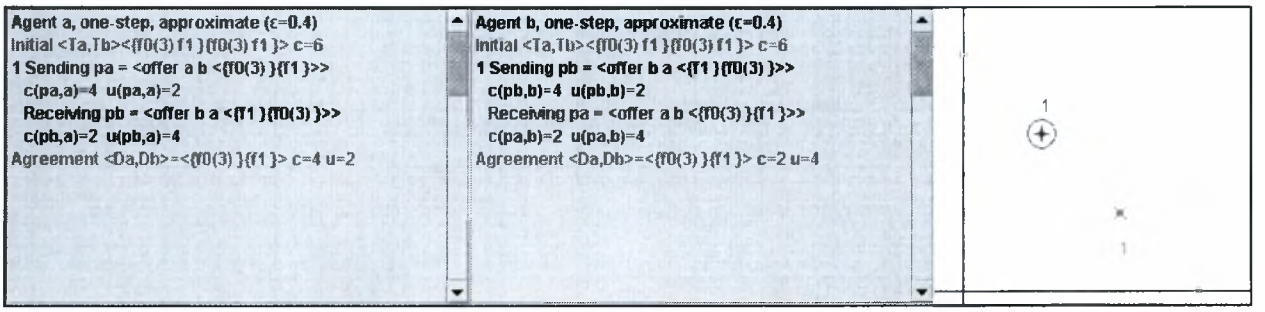
Σενάριο 3, ΠΕΒ, ΨΜΑ.



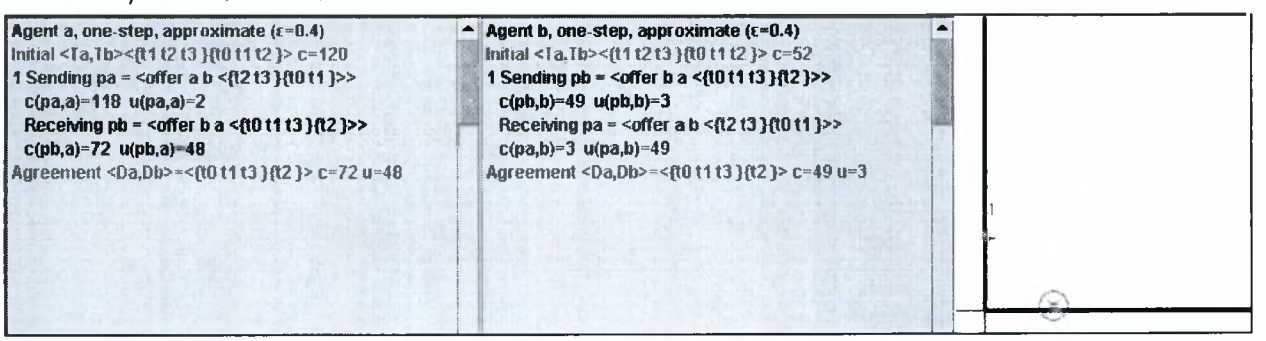
Με χρήση της ΠΜΨ, το τελικό αποτέλεσμα της διαπραγμάτευσης με ΠΕΒ είναι το ίδιο ή καλύτερο από αυτό της διαπραγμάτευσης με ΜΠΣ & Zeuthen και μάλιστα προκύπτει σε ένα μόνο γύρο. Ειδικότερα, στο σενάριο 4 το αποτέλεσμα είναι το ίδιο με αυτό της διαπραγμάτευσης με ΜΠΣ & Zeuthen. Στο σενάριο 3.1 το αποτέλεσμα είναι βέλτιστο, ενώ στη διαπραγμάτευση με ΜΠΣ & Zeuthen είχε προκύψει $<\{t_2,t_3\}\{t_0,t_1\}>$ με $cost_a = 118$ και $cost_b = 3$.

Η βελτίωση αυτή με το ΠΕΒ παρατηρείται γιατί η ΠΜΨ εντοπίζει μια περιορισμένη λίστα υποψήφιων προτάσεων και όχι ολόκληρο το σύνολο διαπραγμάτευσης, όπως οι ΕΜΑ και ΨΜΑ. Στην περίπτωση αυτή, κάθε πράκτορας ελέγχει όλες τις προτάσεις που εντοπίστηκαν και επιλέγει την καλύτερη. Από τις καλύτερες για κάθε πράκτορα επιλέγεται η βέλτιστη για το σύστημα. Αντίθετα, στο ΜΠΣ παράγεται και αποστέλεται μια-μια πρόταση και συγκρίνεται κάθε φορά με την πρόταση του αντιπάλου. Έτσι, μπορεί το αποτέλεσμα να είναι χειρότερο για το σύστημα.

Σενάριο 4, ΠΕΒ, ΠΜΨ.

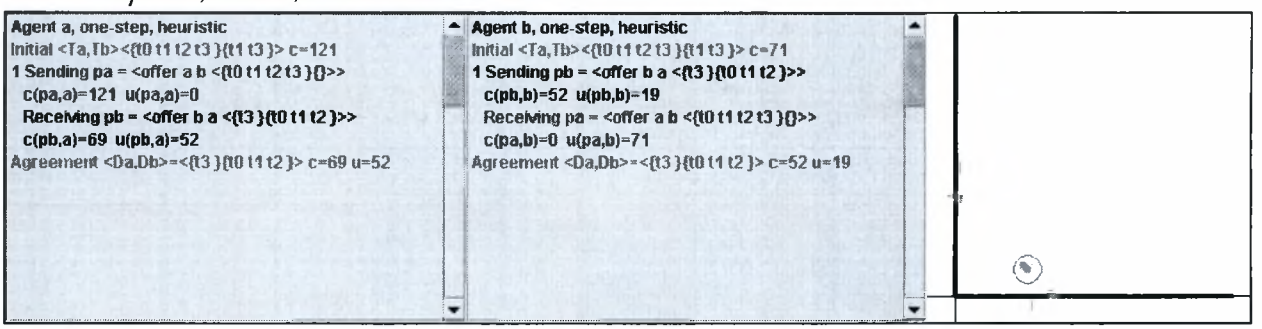


Σενάριο 3.1, ΠΕΒ, ΠΜΨ.

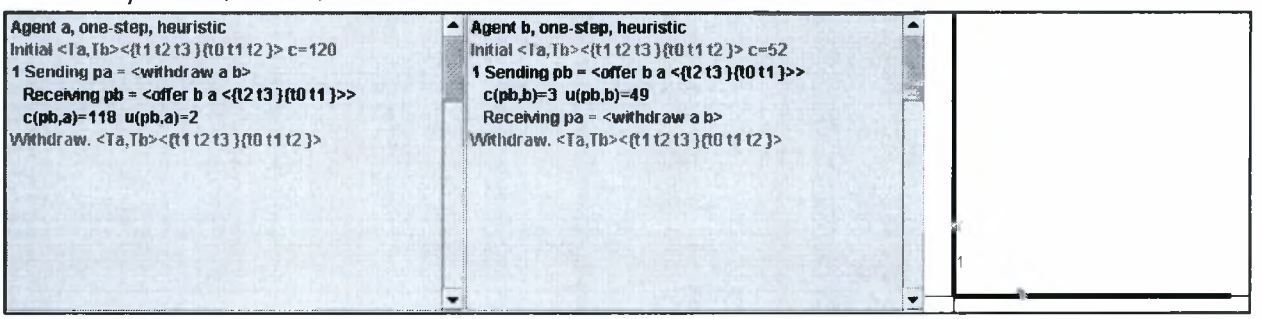


Με χρήση της ΕΜ, το τελικό αποτέλεσμα της διαπραγμάτευσης με ΠΕΒ είναι το ίδιο ή καλύτερο από αυτό της διαπραγμάτευσης με ΜΠΣ & Zeuthen και μάλιστα προκύπτει σε ένα μόνο γύρο. Στο σενάριο 3 το αποτέλεσμα είναι βέλτιστο, ενώ στη διαπραγμάτευση με ΜΠΣ & Zeuthen είχε προκύψει $<\{t_2,t_3\}\{t_0,t_1\}>$ με $cost_a = 118$ και $cost_b = 3$. Στο σενάριο 3.1 το αποτέλεσμα είναι το ίδιο με αυτό της διαπραγμάτευσης με ΜΠΣ & Zeuthen. Στο σενάριο 5 το αποτέλεσμα είναι επίσης το ίδιο με αυτό της διαπραγμάτευσης με ΜΠΣ & Zeuthen.

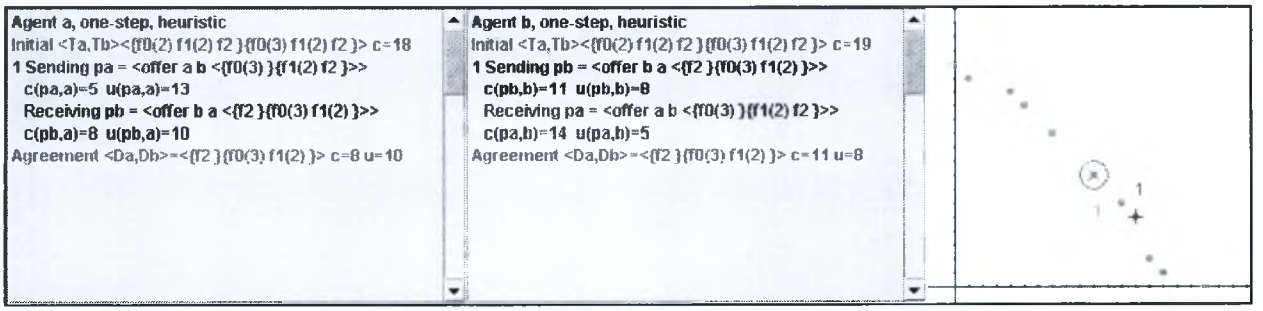
Σενάριο 3, ΠΕΒ, ΕΜ.



Σενάριο 3.1, ΠΕΒ, ΕΜ.

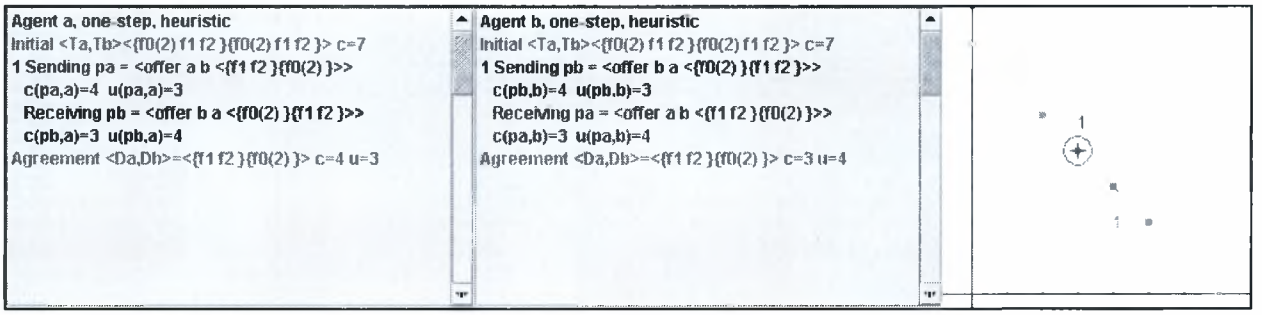


Σενάριο 5, ΠΕΒ, ΕΜ.

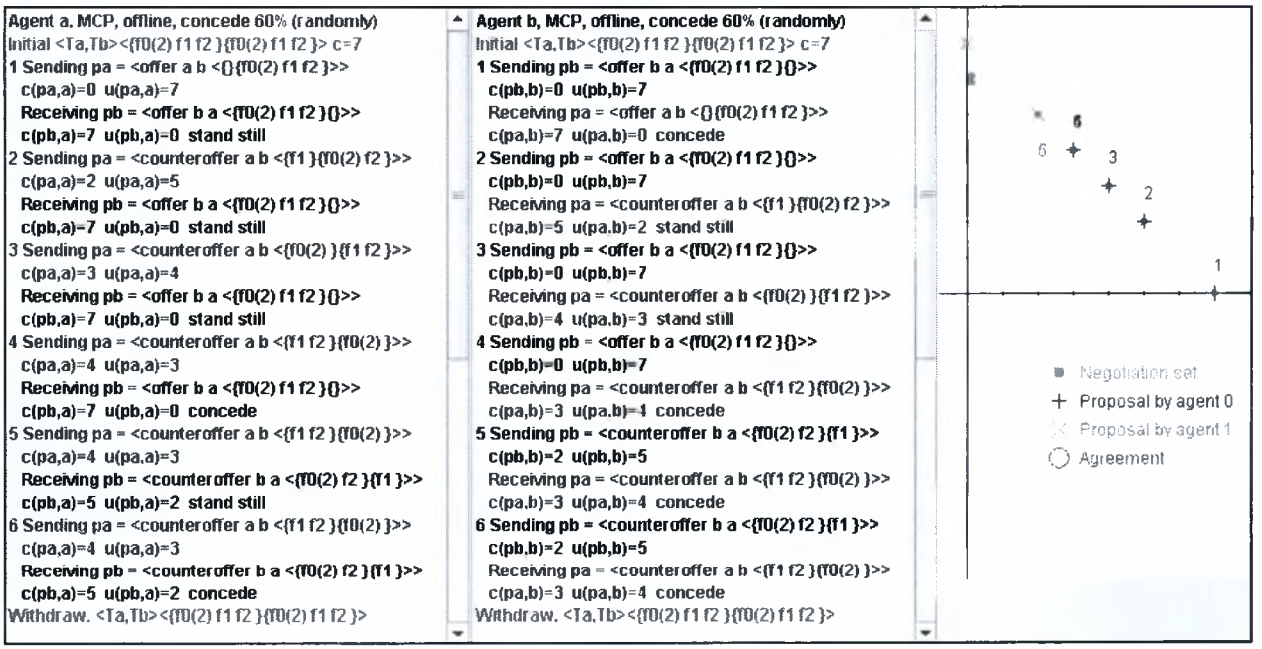


Στο σημείο αυτό πρέπει να κάνουμε την εξής παρατήρηση. Έχουμε διαπιστώσει ότι η χρήση ΜΠΣ & Zeuthen παράγει βέλτιστα αποτελέσματα συγκριτικά με το ΜΠΣ με άλλες πολιτικές υποχώρησης. Επομένως, δεν χρειάζεται καν να συγκρίνουμε πειραματικά τα αποτελέσματα του ΠΕΒ με αυτά του ΜΠΣ με τις πολιτικές 2, 3 και 4: θα είναι σίγουρα καλύτερα ή το πολύ τα ίδια. Ενδεικτικά αναφέρεται το επόμενο σενάριο.

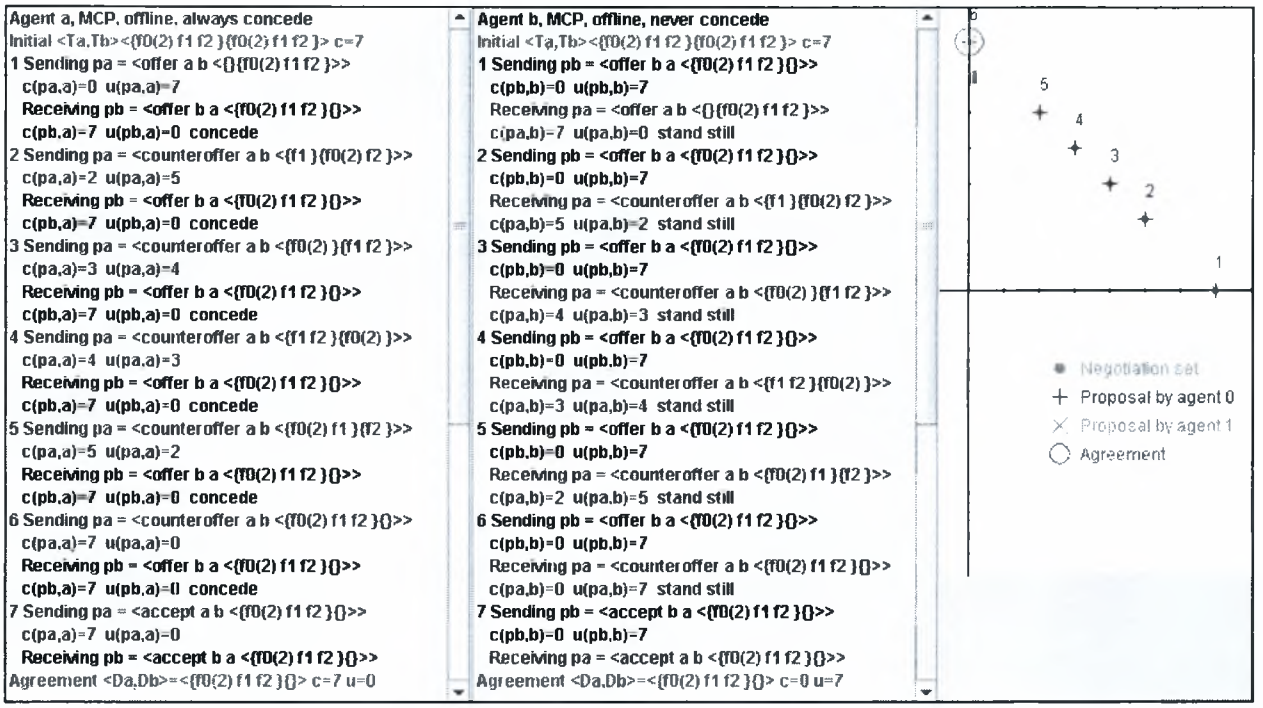
Σενάριο 2, ΠΕΒ, ΕΜ.



Σενάριο 2, ΜΠΣ & τυχαία υποχώρηση, ΕΜ.



Σενάριο 2, ΜΠΣ & συνεχής – μηδενική υποχώρηση, ΕΜ.



Αποτίμηση – σύγκριση πρωτοκόλλων.

Καταρχήν, τόσο στο ΜΠΣ, όσο και στο ΠΕΒ, το υπολογιστικό κόστος είναι παρόμοιο, γιατί η διαδικασία offline υπολογισμού είναι η ίδια ακριβώς. Στο ΠΕΒ επιπλέον υπολογίζονται και συγκρίνονται τα γινόμενα των χρησιμοτήτων, ενώ στο ΜΠΣ υπολογίζεται το ρίσκο όταν χρησιμοποιείται η στρατηγική Zeuthen. Στο ΠΕΒ, όμως, έχουμε μικρότερο επικοινωνιακό κόστος, εφόσον η διαπραγμάτευση διαρκεί ένα βήμα μόνο.

Στη συνέχεια, υπενθυμίζεται ότι μπορεί να γίνει χρήση των τεσσάρων μεθόδων για τον προσδιορισμό των υποψήφιων προτάσεων. Από τα σενάρια που μελετήθηκαν επαληθεύεται ότι για τις ΕΜΑ και ΨΜΑ που υπολογίζουν όλες τις προτάσεις του συνόλου διαπραγμάτευσης, τόσο στο ΜΠΣ με τη στρατηγική Zeuthen, όσο και στο ΠΕΒ, η τελική πρόταση είναι pareto βέλτιστη και μεγιστοποιεί το γινόμενο των χρησιμοτήτων των δυο πρακτόρων. Το τελικό αποτέλεσμα είναι το ίδιο και στις δυο περιπτώσεις, εκτός αν υπάρχουν περισσότερες από δυο προτάσεις που μεγιστοποιούν το γινόμενο των χρησιμοτήτων, οπότε και πραγματοποιείται τυχαία επιλογή.

Με τη χρήση των ΠΜΨ και ΕΜ το τελικό αποτέλεσμα με χρήση ΠΕΒ μπορεί να είναι ίδιο ή καλύτερο από τη χρήση ΜΠΣ & Zeuthen. Παρόλ' αυτά, το αποτέλεσμα δεν είναι πάντα pareto βέλτιστο, όπως σχολιάστηκε και στην αποτίμηση των μεθόδων στο κεφάλαιο 4.

Με τη χρήση άλλων πολιτικών υποχώρησης στο ΜΠΣ, το τελικό αποτέλεσμα είναι το ίδιο ή χειρότερο του ΠΕΒ.

Ανακεφαλαίωση.

Στο κεφάλαιο αυτό περιγράφηκε το Πρωτόκολλο ενός Βήματος και δόθηκαν οι διαφορές με το ΜΠΣ ως προς τη σχεδίαση και την υλοποίησή του. Επίσης πραγματοποιήθηκε θεωρητική και πειραματική αποτίμηση του ΠΕΒ και σύγκρισή του με το ΜΠΣ.

Στην παρούσα εργασία σχεδιάστηκε, υλοποιήθηκε και μελετήθηκε ένα απλό σύστημα διαπραγμάτευσης μεταξύ δύο πρακτόρων για ΤΠΕ απλών και σύνθετων εργασιών, με χρήση του μονοτονικού πρωτοκόλλου συμβιβασμού και του πρωτοκόλλου ενός βήματος. Σε ότι αφορά το πρώτο πειραματιστήκαμε με διαφορετικές πολιτικές υποχώρησης και με διαφορετικές μεθόδους εύρεσης των προτάσεων που υποβάλλονται από τους πράκτορες κατά τη διάρκεια της διαπραγμάτευσης.

Το σύστημα που αναπτύχθηκε θα μπορούσε να βελτιωθεί από πλευράς ταχύτητας υλοποίησης, ακόμη και αν ο κώδικας γίνει λιγότερο ευανάγνωστος. Επίσης, θα μπορούσε να γίνει καλύτερη και πιο εντυπωσιακή η γραφική διασύνδεση με το χρήστη. Περισσότερο ουσιαστικές όμως είναι οι ενδεχόμενες βελτιώσεις που αφορούν αυτή καθ'εαυτή τη διαδικασία διαπραγμάτευσης, και σε αυτές εστιάζεται η συζήτηση που ακολουθεί σε αυτό το κεφάλαιο.

1. Κόστος διαπραγμάτευσης

Κάποιες φορές τίθεται το ερώτημα κατά πόσο συμφέρει η διαδικασία της διαπραγμάτευσης. Κι αυτό γιατί, τόσο ο offline υπολογισμός, όσο και η διαπραγμάτευση επιβαρύνουν το σύστημα με υπολογιστικό και επικοινωνιακό κόστος. Όταν βρεθεί συμβιβαστική λύση το τελικό κέρδος είναι ικανοποιητικό. Πρόβλημα υπάρχει όταν γίνει όλη η παραπάνω διαδικασία και τελικά προκύψει σύγκρουση. Μπορεί να αποφευχθεί αυτή η κατάσταση; Μια απλή ιδέα είναι να γίνει από κάθε πράκτορα μια πρώτη σύγκριση του αρχικού και τελικού σημείου διαπραγμάτευσης. Αν το κόστος τελικού σημείου είναι μικρότερο του κόστους αρχικού, τότε η αρχική αλληλεπίδραση είναι η καλύτερη λύση και δεν έχει νόημα η περεταίρω διαπραγμάτευση. Αν αρχικό και τελικό σημείο είναι πολύ κοντά, τότε ίσως η διαπραγμάτευση να μην έχει νόημα. Αν αντίθετα το κόστος αρχικού σημείου είναι πολύ μικρότερο του τελικού, τότε μάλλον θα προκύψει συμβιβαστική λύση ικανοποιητική και για τους δυο.

Το παραπάνω εξαρτάται και από το κόστος εκτέλεσης των εργασιών. Μπορεί να συγκριθεί με κάποιο τρόπο με το κόστος διαπραγμάτευσης; Είναι τελικά για κάθε πράκτορα το κόστος εκτέλεσης εργασιών με βάση την τελική αλληλεπίδραση μαζί με το κόστος διαπραγμάτευσης μικρότερο του κόστους εκτέλεσης εργασιών σύμφωνα με την αρχική αλληλεπίδραση;

2. Πειραματισμός με παραλλαγές των πρωτοκόλλων που υλοποιήθηκαν

Θα είχε ενδιαφέρον να δούμε κάποιες παραλλαγές στα πρωτόκολλα και τις πολιτικές πρακτόρων που εξετάστηκαν. Έτσι, μπορούμε να μελετήσουμε θεωρητικά ή πειραματικά πώς επηρεάζεται η διαδικασία της διαπραγμάτευσης, αν στο ΜΠΣ δεν επιτρέπεται να παραμένει κανείς στην προηγούμενη θέση του. Επιτρέπεται μόνο να υποχωρεί ή να εγκαταλείψει.

Επίσης, είδαμε ότι όταν χρησιμοποιεί στρατηγική Zeuthen κάθε πράκτορας υπολογίζει τις τιμές ρίσκου και, αν βρει το δικό του μικρότερο ή ίσο από του αντιπάλου, η στρατηγική προτείνει να υποχωρήσει ο πράκτορας, διαφορετικά να παραμείνει στη θέση του. Ένας ριψοκίνδυνος πράκτορας θα μπορούσε να αρνηθεί συμμόρφωση με τη στρατηγική στην περίπτωση ισότητας στην παραπάνω σχέση, ρισκάροντας όμως σύγκρουση, αν δεν υποχωρήσει και ο αντίπαλος. Εναλλακτικά, θα μπορούσε να επιλέξει με τυχαίο τρόπο, αν θα υποχωρήσει ή όχι. Βέβαια, η πολιτική αυτή μπορεί να οδηγήσει σε γρηγορότερη σύγκλιση, αλλά δεν είναι πάντα αποτελεσματική. Θα είχε όμως ενδιαφέρον να μελετήσουμε τις διαφορές στην πορεία της διαπραγμάτευσης υπό τις παραπάνω συνθήκες.

3. Δυναμική επιλογή μεθόδου εύρεσης επόμενης πρότασης

Μια άλλη ιδέα είναι η επιλογή της χρησιμοποιούμενης μεθόδου στη διαπραγμάτευση να μη γίνεται από το χρήστη, αλλά από τον πράκτορα ή το σύστημα. Για παράδειγμα, ο χρήστης μπορεί απλά να επιλέξει ότι τον ενδιαφέρει μια λύση ακρίβειας και το πρόγραμμα να κάνει επιλογή μεταξύ της μεθόδου ΕΜΑ τάξης $O(2^N)$ και της μεθόδου ΨΜΑ τάξης $O(NW)$. Αν, λοιπόν, διαπιστώσουμε ότι $2^N >> NW$, τότε συμφέρει η ΨΜΑ. Αν ισχύει $2^N << NW$, συμφέρει η ΕΜΑ.

4. Άλλες μέθοδοι εύρεσης επόμενης πρότασης

Η μέθοδος ΕΜ είδαμε ότι σε κάθε γύρο βασίζεται στην επιλογή της κατάλληλης με κάποιο κριτήριο εργασίας από τον αντίπαλο. Ως κατάλληλη θεωρήσαμε μέχρι στιγμής την εργασία μικρότερου κόστους. Θα είχε ενδιαφέρον να μελετήσουμε και άλλα κριτήρια, όπως την πρώτη στη σειρά, τη μεγαλύτερου κόστους ή αυτή που προκύπτει από τυχαία επιλογή. Επίσης, μπορούμε να δοκιμάσουμε να επιλέξουμε σε κάθε βήμα μια εργασία από τον αντίπαλο και να

του επιστρέψουμε μια άλλη, έτσι ώστε να πληρούνται πάντα οι προϋποθέσεις του ΜΠΣ.

Μπορούμε να βρούμε και να μελετήσουμε και άλλες μεθόδους για την εύρεση επόμενης πρότασης σ' ένα γύρο της διαπραγμάτευσης. Είδαμε ότι το πρόβλημα αυτό παρουσιάζει ομοιότητες με το subset sum και με το knapsack και μπορούμε να πάρουμε ιδέες από τις λύσεις που προτείνονται στη βιβλιογραφία για αυτά τα προβλήματα.

Ενδεικτικά, παρουσιάζουμε άλλη μια προσεγγιστική μέθοδο βασισμένη στην τεχνική της απληστίας (*greedy strategy*). Οι *άπληστοι αλγόριθμοι*[32] κάνουν μια επιλογή που δείχνει βέλτιστη τη δεδομένη στιγμή και ελπίζουν ότι αυτή θα οδηγήσει σε ολικά βέλτιστη λύση. Χρησιμοποιούμε τη λύση που προτείνεται σε ανάλογο πρόβλημα στο [Martello 1990], την προσαρμόζουμε και καταλήγουμε στην εξής πρόταση: Αρχικά το σύνολο εργασιών ταξινομείται κατά φθίνουσα σειρά κόστους με πολυπλοκότητα O(NlogN). Για να βρει ο πράκτορας την επόμενη πρόταση κόστους w = αρχικό σημείο, αρχικό+1, ..., τελικό σημείο, προσθέτει μια-μια εργασία ξεκινώντας από τη μεγαλύτερη που χωράει, έτσι ώστε το τρέχον άθροισμα να είναι μεγαλύτερο ή ίσο του κόστους αρχικού σημείου και μικρότερο ή ίσο του w. Αν W' = τελικό – αρχικό, η διαδικασία έχει στη χειρότερη περίπτωση κόστος O(W'N). Η διαδικασία μπορεί να πραγματοποιηθεί offline ή μη. Στη δεύτερη περίπτωση θα μπορούσε να μειωθεί η πολυπλοκότητα. Κι αυτό γιατί αν ο υπολογισμός γίνεται σε κάθε γύρο τότε μπορεί η λύση να προκύψει πολύ πριν το τελικό σημείο και η πολυπλοκότητα θα είναι O(W"N) < O(W'N), με W" = w - αρχικό σημείο.

Μια παραλλαγή είναι η εξής: να μην αντιμετωπίζει η μέθοδος όλα τα w, αλλά να προσπερνάει κάποια. Για παράδειγμα αν το τρέχον άθροισμα είναι 20 και η συμφερότερη εργασία έχει κόστος 5, τότε να προσπερνάει τα w = 21, 22, 23, 24 και να θέσει κατευθείαν w = 25.

5. Πειραματισμός με άλλα είδη εργασιών και τομέων

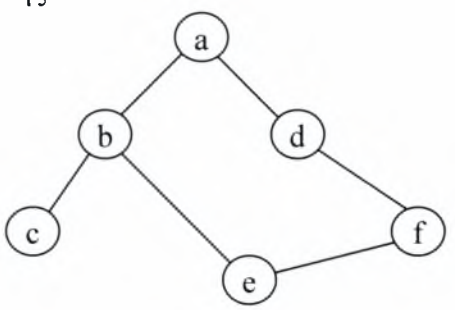
Μέχρι στιγμής αναφερθήκαμε σε απλές και σύνθετες εργασίες, όπως ορίστηκαν στην παράγραφο 2.1 και προτείναμε ενιαία διαχείρισή τους. Με τον τρόπο αυτό καλύπτεται ένας μεγάλος αριθμός προβλημάτων, όπως ο τομέας αποστολής φαξ ή αναζήτησης σε βάση δεδομένων. Έχει νόημα να δούμε αν υπάρχουν άλλα ήδη εργασιών και τομέων και το πώς θα μπορούσαν να αντιμετωπιστούν.

---

[32] [Cormen 2001 κεφ.16]

Αν καταρχήν, η συνάρτηση κόστους εξαρτάται μεν μόνο από την ποσότητα κάθε υποεργασίας, αλλά όχι απαραίτητα με μια σχέση γραμμική όπως αυτής της παραγράφου 2.1, αλλάζουν λίγα πράγματα μόνο στο τμήμα της μετατροπής των σύνθετων εργασιών σε υποεργασίες και ειδικότερα κατά τον υπολογισμό του κόστους υποεργασιών. Στη συνέχεια χρησιμοποιείται κανονικά η υπάρχουσα υποδομή.

Ένα άλλο ζήτημα είναι οι εργασίες που παρουσιάζουν αρκετές εξαρτήσεις μεταξύ τους (interdependent tasks). Για παράδειγμα, πώς μπορούμε μέσα από το υπάρχον σύστημα να αντιμετωπίσουμε προβλήματα όπως του τομέα της αλληλογραφίας ή της παράδοσης που περιγράφηκαν στο υποκεφάλαιο 1.4; Όπως φαίνεται στο σχήμα 6.1, οι διαδρομές και κατά συνέπεια οι εργασίες παρουσιάζουν αλληλεξαρτήσεις. Π.χ. η διαδρομή ac περιλαμβάνει τη διαδρομή ab, δηλαδή ένας πράκτορας που έχει να παραδώσει γράμματα στον κόμβο c μπορεί χωρίς επιπλέον κόστος να παραδώσει και στον κόμβο b. Η εργασία ac μπορεί να σπάσει στις ab και bc που είναι δευτερεύουσα της ab. Επομένως, αντί για τις ab και ac, έχουμε τις ab και ab,bc στις οποίες η επικάλυψη που παρουσιάζεται είναι προφανής.



Σχήμα 6.1

Ας παρακολουθήσουμε μια διαπραγμάτευση για το παραπάνω παράδειγμα. Έστω αρχική αλληλεπίδραση e = <{ab}{ab,bc}>. Για τον πράκτορα 0 υποψήφιες προτάσεις είναι <{}{ab,bc}>, <{ab}{ab,bc}>, <{ab,bc}{}>. Η <{ab}{ab,bc}> απορρίπτεται γιατί η <{}{ab,bc}> είναι καλύτερη για τον πράκτορα χωρίς να είναι χειρότερη για τιν αντίπαλο και την αντικαθιστά. Η <{ab,bc}{}> απορρίπτεται γιατί είναι εκτός του ορίου του τελικού σημείου. Άρα στο σύνολο διαπραγμάτευσης απομένει μόνο η <{}{ab,bc}>. Για τον πράκτορα 1 υποψήφιες προτάσεις είναι <{ab,bc}{}>, <{ab,bc}{ab}>, <{}{ab,bc}>, από τις αποίες απομένει η <{}{ab,bc}>. Οι δυο πράκτορες κάνουν την ίδια πρώτη πρόταση και καταλήγουν σε συμφωνία.

Στην περίπτωση που παρουσιάζονται πολλαπλές εξαρτήσεις μεταξύ των εργασιών και υποεργασιών αυτές πρέπει να εισάγονται στο σύστημα και να αποθηκεύονται κάπου. Αυτό πραγματοποιείται όπως είδαμε στο υποκεφάλαιο

3.3 'Αναπαράσταση της πληροφορίας', αν στον πίνακα dependency αποθηκεύεται όχι μόνο ένας κωδικός (της κύριας εργασίας), αλλά ο δεκαδικός ακέραιος, που αντιστοιχεί στο δυαδικό αριθμό, που αντιστοιχεί με τη σειρά του μοναδικά σ' ένα υποσύνολο εργασιών από τις οποίες εξαρτάται. Π.χ 14 ↔ 1110 ↔ {t3,t2,t1}. Αν λοιπόν αποθηκευτούν από την αρχή οι εξαρτήσεις μεταξύ των εργασιών μπορεί και πάλι να χρησιμοποιηθεί η υπάρχουσα υποδομή.

Γενικά σε κάθε τομέα που συναντάμε πρέπει ο χρήστης να κατανοήσει σε βάθος το πρόβλημα, να κάνει κάποια προεργασία και να ορίσει καλά στο σύστημα τις υπάρχουσες εργασίες, το κόστος και τις εξαρτήσεις που παρουσιάζουν.

Από τη βιβλιογραφία, οι ΤΠΕ κατηγοριοποιούνται[33] με βάση κάποιες διαφορετικές ιδιότητες που έχει η συνάρτηση κόστους.

Ένας ΤΠΕ θα λέγεται subadditive αν για όλα τα πεπερασμένα υποσύνολα εργασιών $X,Y \subseteq T$, ισχύει $cost(X \cup Y) \leq cost(X) + cost(Y)$. Με άλλα λόγια, ένας τομέας δεν είναι subadditive αν μια εργασία έχει άλλο κόστος όταν εκτελείται μόνη της και άλλο όταν εκτελείται μαζί με άλλη. Για παράδειγμα, το fax domain και το postmen domain είναι subadditive.

Ένας ΤΠΕ θα λέγεται concave αν για όλα τα πεπερασμένα υποσύνολα εργασιών $X \subseteq Y$, $Z \subseteq T$, ισχύει $cost(Y \cup Z) - cost(Y) \leq cost(X \cup Z) - cost(X)$. Με άλλα λόγια, ένας τομέας δεν είναι concave αν μια εργασία ή υποσύνολο εργασιών μπορεί να προσθέσει περισσότερο κόστος σε ένα σύνολο από ότι σε ένα υποσύνολό του. Για παράδειγμα, το fax domain είναι concave.

Ένας ΤΠΕ θα λέγεται modular αν για όλα τα πεπερασμένα υποσύνολα εργασιών $X,Y \subseteq T$, ισχύει $cost(X \cup Y) = cost(X) + cost(Y) - cost(X \cap Y)$. Για παράδειγμα, το fax domain είναι modular.

Εκτός από τους ΤΠΕ υπάρχουν οι State Oriented Domains (SOD) και Worth Oriented Domains (WOD), που περιγράφονται στο [Rosenschein 1994].

Θα είχε ενδιαφέρον όλα τα παραπάνω να εξεταστούν καλύτερα και να δούμε αν ο τρόπος αντιμετώπισης που προτείνεται στην παρούσα εργασία επαρκεί για όλες τις περιπτώσεις και αν όχι τι πρέπει να γίνει. Επίσης, μπορούμε να δούμε αν είναι δυνατό σε κάποιες περιπτώσεις να γίνουν απλοποιήσεις στις υποθέσεις που έχουν γίνει στο σύστημα έτσι ώστε να γίνει κατά περίπτωση ταχύτερο.

[33] [Rosenschein 1994]

6. Διαπραγμάτευση μεταξύ ανειλικρινών πρακτόρων

Μέχρι τώρα υποθέσαμε ότι οι πράκτορες λένε την αλήθεια. Αυτό όμως δεν ισχύει πάντα, στην πράξη μπορεί να πουν ψέματα ως προς τις εργασίες που έχουν αναλάβει κατά την αρχική αλληλεπίδραση, έτσι ώστε να ανατρέψουν την πορεία της διαπραγμάτευσης. Αυτό μπορεί να επιτευχθεί με δυο τρόπους:

- δήλωση εργασίας φαντάσματος (phantom) ή δολώματος (decoy), δηλαδή προσποίηση ότι έχει γίνει ανάθεση μιας εργασίας, χωρίς αυτό να ισχύει. Για παράδειγμα, στο σχήμα 6.1 του postmen domain, αν και οι δυο πράκτορες έχουν να επισκεφτούν μόνο τον κόμβο d, τότε η εργασία μπορεί να ανατεθεί είτε στον ένα, είτε στον άλλο, με 50% πιθανότητα στον καθένα. Αν κάποιος δηλώσει ψευδώς ότι έχει να παραδώσει επίσης γράμμα στον κόμβο c, τότε ο κόμβος d θα ανατεθεί στον αντίπαλο και ο ίδιος τελικά δεν θα κάνει τίποτα. Για την αποφυγή τέτοιων περιπτώσεων, μπορεί ο πράκτορας που υποψιάζεται τον αντίπαλο να ζητήσει απόδειξη για την ύπαρξη της εργασίας. Αν είναι δυνατή η παραγωγή μιας τεχνητής ψεύτικης εργασίας, τότε αναφερόμαστε σε decoy εργασίες, αλλιώς σε phantom.

- απόκρυψη εργασίας (hidden task): κατά ανάλογο τρόπο, μπορεί να συμφέρει τον πράκτορα να αποκρύψει την ύπαρξη μιας εργασίας. Για παράδειγμα, στο postmen domain, αν ο πράκτορας A έχει να παραδώσει γράμμα στους κόμβους b και d και ο B μόνο στον d, τότε μετά από διαπραγμάτευση θα προέκυπτε να παραδώσει ο A στους b και d και ο B να μην κάνει τίποτα. Αν όμως ο A αποκρύψει το γράμμα για τον b, τότε ενδέχεται κατά 50% να ανατεθεί στον B η εργασία, οπότε στον A θα απομείνει μόνο ο κόμβος d. Ακόμη όμως και αν ανατεθεί σε αυτόν ο κόμβος b δεν έχει να χάσει κάτι σε σχέση με την αρχική αλληλεπίδραση ή σε σχέση με την τελική συμφωνία σε περίπτωση ειλικρίνειας.

Πως εξελίσσεται η διαπραγμάτευση αν οι πράκτορες ψεύδονται και πώς μπορεί κάποιος να σιγουρευτεί για τον αντίπαλο; Με τα ζητήματα σχετικά με τα ψεύδη ασχολείται το [Rosenschein 1994].

7. Διαπραγμάτευση μεταξύ περισσοτέρων από δύο πρακτόρων

Μέχρι τώρα θεωρήσαμε ότι η διαπραγμάτευση αφορά 2 πράκτορες, δηλαδή είναι διμερής (bilateral). Τι γίνεται όταν υπάρχουν K πράκτορες που διαπραγματεύονται ταυτόχρονα μεταξύ τους; Στην πολύπλευρη (multilateral) διαπραγμάτευση[34], τα πράγματα γίνονται ιδιαίτερα πολύπλοκα.

[34] [Lomuscio 2000], [Endriss 2006]

Αναφορικά με το πρωτόκολλο ενός βήματος, η διαδικασία είναι η εξής: κάθε πράκτορας προτείνει την πρόταση που μεγιστοποιεί το γινόμενο των χρησιμοτήτων και τον συμφέρει περισσότερο. Από τις Κ προτάσεις επιλέγεται αυτή που μεγιστοποιεί το γινόμενο των χρησιμοτήτων. Το πρωτόκολλο είναι σταθερό και αποτελεσματικό.

Αναφορικά με το μονοτονικό πρωτόκολλο συμβιβασμού, στον πρώτο γύρο κάθε πράκτορας κάνει μια πρόταση. Σε κάθε επόμενο γύρο κάθε πράκτορας είτε υποχωρεί είτε παραμένει στη θέση του μέχρι να επέλθει συμφωνία ή σύγκρουση. Στη συνέχεια θα οριστούν λίγο καλύτερα οι έννοιες της υποχώρησης και συμφωνίας στην περίπτωση αυτή. Συμφωνία επέρχεται αν κάποιος πράκτορας κάνει μια πρόταση που είναι τουλάχιστο τόσο καλή για κάθε άλλο πράκτορα όσο και η δική του. Η υποχώρηση από την άλλη είναι πιο πολύπλοκη και μπορεί να είναι κάποια από τις παρακάτω: ισχυρή υποχώρηση όταν η νέα πρόταση είναι καλύτερη για όλους τους αντιπάλους, ασθενής υποχώρηση όταν η νέα πρόταση είναι καλύτερη για ένα τουλάχιστο αντίπαλο, pareto υποχώρηση όταν η νέα πρόταση είναι τουλάχιστο το ίδιο καλή για όλους τους αντιπάλους και καλύτερη για ένα τουλάχιστο αντίπαλο, κοκ. Στην περίπτωση αυτή η εύρεση υποψήφιων προτάσεων μπορεί να είναι της τάξης του K^N αντί για 2^N.

Μια διαφορετική ιδέα είναι σε κάθε γύρο ο πράκτορας να προτείνει κάποιες εργασίες για τον εαυτό του και να αφήνει τους αντιπάλους να διαπραγματευτούν μεταξύ τους για τις υπόλοιπες εργασίες.

Το πρωτόκολλο Contract Net (CNET)[35] (CNET) είναι ένα υψηλού επιπέδου πρωτόκολλο για την επίτευξη αποτελεσματικής συνεργασίας μέσω μοιράσματος εργασιών, σύμφωνα με την ακόλουθη διαδικασία: α) Ένας πράκτορας, που λέγεται διαχειριστής, παράγει μια εργασία και ανακοινώνει την ύπαρξή της σε όλους τους διαθέσιμους πράκτορες ή επιλεκτικά σε κάποιους (task announcement). β) Οι πράκτορες που ακούν την ανακοίνωση της εργασίας, την αποτιμούν με βάση τους διαθέσιμους πόρους και, αν κρίνουν ότι ενδιαφέρονται, κάνουν μια προσφορά (bidding). γ) Ο διαχειριστής δέχεται τις προσφορές από κάθε ανακοίνωση εργασίας, επιλέγει τους πιο κατάλληλους ενδιαφερόμενους πράκτορες που πλέον λέγονται εργολάβοι (contractors) και τους ενημερώνει με ένα μήνυμα απονομής (awarding the contract).

Αξίζει τον κόπο να μελετηθούν οι ομοιότητες και διαφορές του παραπάνω με το ΜΠΣ για Κ πράκτορες.

---

[35] [Smith 1981]

8. Διαπραγμάτευση βασισμένη στη λογική επιχειρηματολογία

Στην παρούσα εργασία μελετήθηκαν τεχνικές διαπραγμάτευσης βασισμένες στη θεωρία παιγνίων και σε ευρετικές τεχνικές. Υπάρχει μια ακόμη τεχνική διαπραγμάτευσης, η βασισμένη σε επιχειρήματα, που είναι πιο πολύπλοκη αλλά παρουσιάζει πολλές δυνατότητες. Θα είχε ενδιαφέρον να συγκριθεί με τα παραπάνω από πλευράς αποτελεσματικότητας, απλότητας, κοκ. Η διαπραγμάτευση βασισμένη σε επιχειρήματα περιγράφεται στη συνέχεια.

Επιχείρημα (argument) είναι ένα τμήμα πληροφορίας που επιτρέπει στον πράκτορα να δικαιολογήσει τις απόψεις του και να επηρεάσει άλλους πράκτορες για την αλήθεια ή το ψεύδος κάποιων θέσεων.[36] Υπάρχουν διάφοροι τύποι επιχειρημάτων: απειλή (threat) ότι η μη αποδοχή θα προκαλέσει κάτι αρνητικό, επιβράβευση (reward) ότι η αποδοχή θα προκαλέσει κάτι θετικό και έκκληση (appeal) ότι ενδείκνυται να προτιμηθεί η πρόταση.

Αν Δ είναι μια βάση δεδομένων, τότε ένα λογικό επιχείρημα πάνω στη Δ είναι ένα ζεύγος <φ,Γ>, όπου φ είναι ένας τύπος, γνωστός ως συμπέρασμα, και $\Gamma \subseteq \Delta$, δηλαδή ένα υποσύνολο του Δ, γνωστό ως βάση (grounds/support), τέτοιο ώστε $\Gamma \vdash \varphi$, δηλαδή το φ προκύπτει από το Γ.[37] Έστω <φ1,Γ1> και <φ2,Γ2> δυο επιχειρήματα για κάποια βάση δεδομένων Δ. Το <φ1,Γ1> μπορεί να *νικήσει* (defeat) το <φ2,Γ2> με έναν από τους εξής τρόπους: Πρώτον, το <φ1,Γ1> αντικρούει (rebut) το <φ2,Γ2>, αν το φ1 επιτίθεται στο φ2. Δεύτερον, το <φ1,Γ1> κόβει (undercuts) το <φ2,Γ2>, αν το φ1 επιτίθεται στο ψ, για κάποιο $\psi \in \Gamma 2$. Το "επιτίθεται" (attack) ορίζεται με τη σειρά του ως εξής: για δυο προτάσεις φ και ψ, λέμε ότι η φ επιτίθεται στην ψ αν και μόνο αν $\varphi \equiv \neg\psi$. Γενικά, η αντίκρουση επιχειρημάτων είναι πιο ισχυρή από το κόψιμό τους.

Κατά τη *διαπραγμάτευση βασισμένη σε επιχειρήματα* (argumentation-based negotiation – ABN) πραγματοποιείται διάλογος μεταξύ των πρακτόρων. Ο ένας πράκτορας στέλνει μια πρόταση με επιχείρημα για την οποία ο αντίπαλος προσπαθεί να νικήσει, δηλαδή να αντικρούσει ή να κόψει, το επιχείρημα του πρώτου με αντεπιχείρημα. Ο πρώτος μπορεί στη συνέχεια να προσπαθήσει να νικήσει το αντεπιχείρημα με ένα νέο επιχείρημα, κοκ. Αυτός που δεν βρίσκει κάτι να απαντήσει είναι υποχρεωμένος να δεχτεί την πρόταση, ενώ ο αντίπαλος είναι ο νικητής.

---

[36] [Jennings et al. 1998]
[37] [Wooldridge 2002, κεφ. 7]

Ανακεφαλαίωση

Στο κεφάλαιο αυτό αναφερθήκαμε σε κάποια ανοιχτά ζητήματα που υπάρχουν στο χώρο αυτό και που θα μπορούσαν να αποτελέσουν βελτιώσεις της παρούσας αλλά και αντικείμενο μελοντικής εργασίας. Ειδικότερα, αναφερθήκαμε στο κόστος που έχει η ίδια η διαδιακασία της διαπραγμάτευσης. Προτείναμε πειραματισμό με παραλλαγές των πρωτοκόλλων που υλοποιήθηκαν. Εξετάσαμε τη δυνατότητα δυναμικής επιλογής της μεθόδου υπολογισμού προτάσεων. Σχολιάσαμε άλλες μεθόδους υπολογισμού προτάσεων. Προτείναμε πειραματισμό με άλλα είδη εργασιών και τομέων. Αναφερθήκαμε στη διαπραγμάτευση μεταξύ ανειλικρινών πρακτόρων, μεταξύ περισσοτέρων από δύο πρακτόρων και στη διαπραγμάτευση βασισμένη στη λογική επιχειρηματολογία.

7. Βιβλιογραφία

[Bergner 1997] K.Bergner, A.Rausch, M.Sihling, *Using UML for Modeling a Distributed Java Application*, 1997.

[Black 2005] Paul E. Black, *Knapsack problem* in Dictionary of Algorithms and Data Structures [online], 2005. http://www.nist.gov/dads/HTML/knapsackProblem.html

[Cadenhead 2003] R.Cadenhead & L.Lemay, *Πλήρες εγχειρίδιο της Java 2*, Εκδ. Γκιούρδας, Αθήνα 2003.

[Cormen 2001] T.Cormen, C.Leiserson, R.Rivest, *Introduction to Algorithms*, MIT Press 2001.

[Durfee 1999] E.H.Durfee, *Distributed Problem Solving and Planning*, Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence, Chapter 3, MIT Press, Cambridge, 1999.

[Eckstein 1998] R. Eckstein, M.Loy, D.Wood, *Java Swing*, O'Reilly 1998.

[Endriss 2006] U.Endriss, *Monotonic Concession Protocols for Multilateral Negotiation*, Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent, 2006.

[Faratin 2000] P.Faratin, *Automated Service Negotiation Between Autonomous Computational Agents*, 2000.

[Flanagan 1997] D.Flanagan, *Java in a Nutshell*, O'Reilly 1997.

[Grand 1997] M.Grand, *Java Language Reference*, O'Reilly 1997.

[Jennings 1998] N.R.Jennings, K.Sycara, M.Wooldridge, *A Roadmap for Agent Research and Development*, 1998.

[Jennings 2001] N.Jennings, P.Faratin, A.Lomuscio, S.Parsons, C.Sierra, M.Wooldridge, *Automated Negotiation: Prospects, Methods and Challenges,* 2001.

[Lomuscio 2000] A.R.Lomuscio, M.Wooldridge & N.Jennings, *A classification scheme for negotiation in electronic commerce*, 2000.

[Martello 1990] S.Martello & P.Toth, *Knapsack problems: Algorithms and Computer Implementations*, John Willey and Sons Ltd, 1990.

[Oaks 1999] S.Oaks & H.Wong, *Java Threads*, 2nd edition, O'Reilly 1999.

[Pisinger 1995] D.Pisinger, *Algorithms for Knapsack problems*, University of Copenhagen, 1995.

[Rahwan 2003a] I.Rahwan, P.McBurney & S.Sonenberg, *Towards a Theory of Negotiation Strategy,* 2003.

[Rahwan 2003b] I.Rahwan, S.Ramchurn, N.Jennings, P.McBurney, S.Parsons, L.Sonenberg, *Argumentation-Based Negotiation*, 2003.

[Rosenschein 1993] J.S.Rosenschein & G.Zlotkin, *Consenting Agents: Designing Conventions for Automated Negotiation,* 1993.

[Rosenschein 1994] J.S.Rosenschein & G.Zlotkin, *Rules of Encounter – Designing Conventions for Automated Negotiation among Computers*, MIT Press 1994.

[Rumbaugh 1999] J.Rumbaugh, I.Jacobson, G.Booch, *The Unified Modeling Language Reference Manual*, Addison-Wesley Longman 1999.

[Smith 1981] R.Smith, *The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver*, 1981

[Wikipedia] Wikipedia, the free encyclopedia, http://en.wikipedia.org/wiki/Main_Page

[Wooldridge 1997] M.Wooldridge & N.Jennings, *The Cooperative Problem Solving Process*, 1997.

[Wooldridge 2002] M.Wooldridge, A*n Introduction to Multiagent Systems*, John Wiley, Chichester 2002.

[Zlotkin 1989] G.Zlotkin & J.S.Rosenschein, *Negotiation and TaskSharing among Autonomous Agents in Cooperative Domains*, 1989.

[Zlotkin 1993a] G.Zlotkin & J.S.Rosenschein, *A Domain Theory for Task Oriented Negotiation*, 1993.

[Zlotkin 1993b] G.Zlotkin & J.S.Rosenschein, *Mechanism Design for Automated Negotiation and its Application to Task Oriented Negotiation*, 1993.

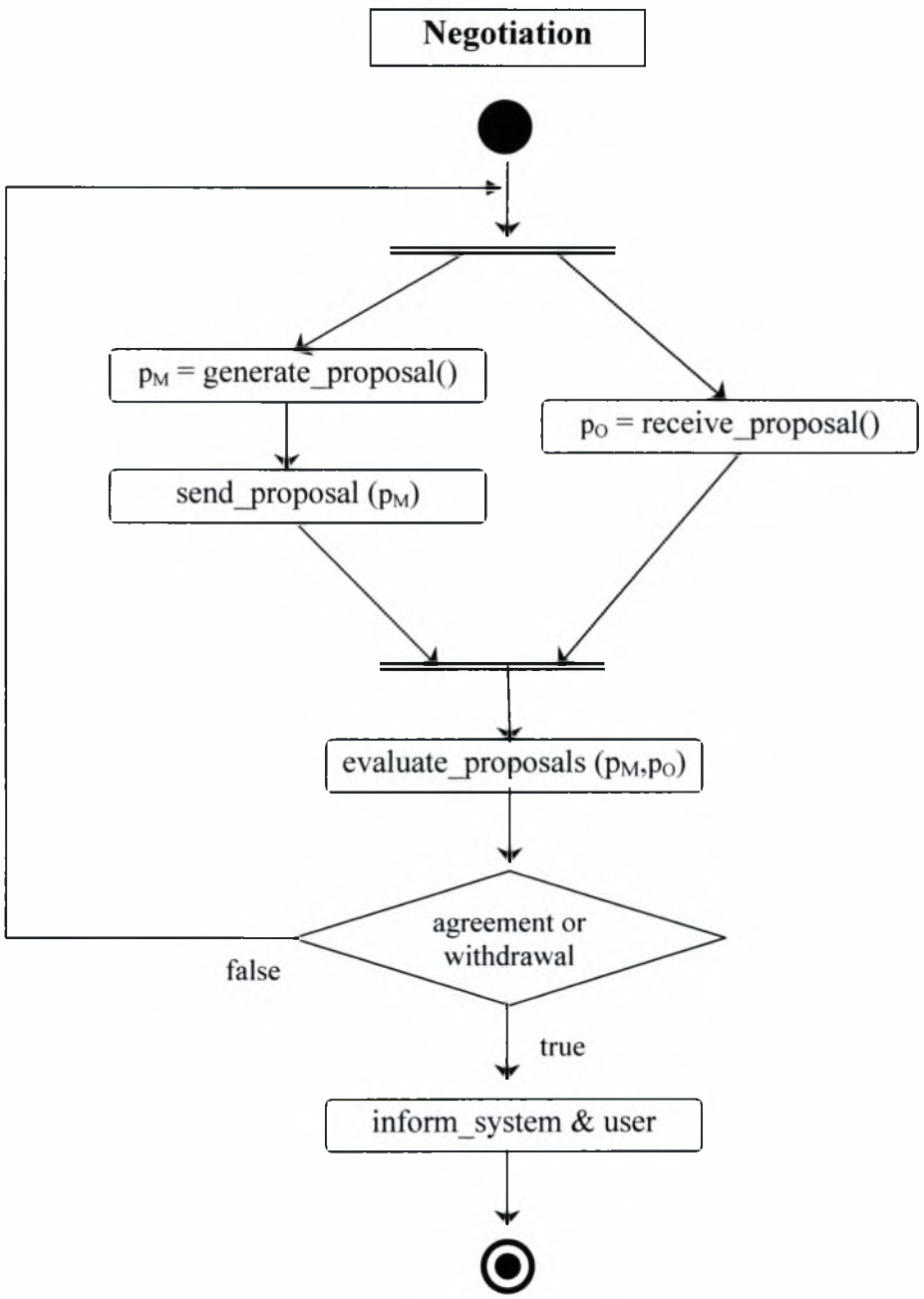
Ακρωνύμια.

ΔΠ	Δυναμικός Προγραμματισμός
ΕΜ	Ευρετική Μέθοδος
ΕΜΑ	Εκθετική Μέθοδος Ακρίβειας
ΜΜΓ	Μηχανισμοί Μεγιστοποίησης Γινομένου
ΜΠΣ	Μονοτονικό Πρωτόκολλο Συμβιβασμού
ΠΕΒ	Πρωτόκολλο Ενός Βήματος
ΠΜΨ	Προσεγγιστική Μέθοδος Ψαλιδίσματος
ΠΣ	Πολυπρακτορικό Σύστημα
ΣΚΕΠ	Συνεργατική Κατανεμημένη Επίλυση Προβλήματος
ΤΠΕ	Τομείς Προσανατολισμένοι σε Εργασίες
ΨΜΑ	Ψευδοπολυωνυμική Μέθοδος Ακρίβειας

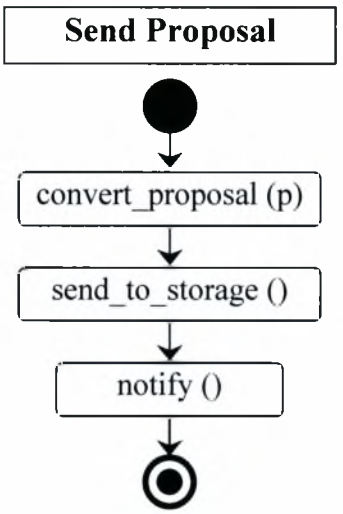
Παράρτημα.

Διαγράμματα δραστηριοτήτων για το σύστημα.

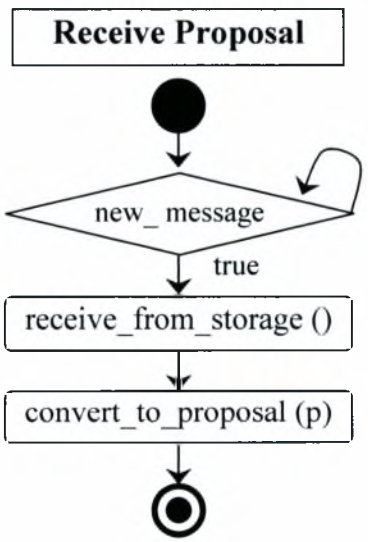
Διάγραμμα Π.1: Δραστηριότητα «Διαπραγμάτευση».
Αναφέρεται στο ΜΠΣ μόνο.



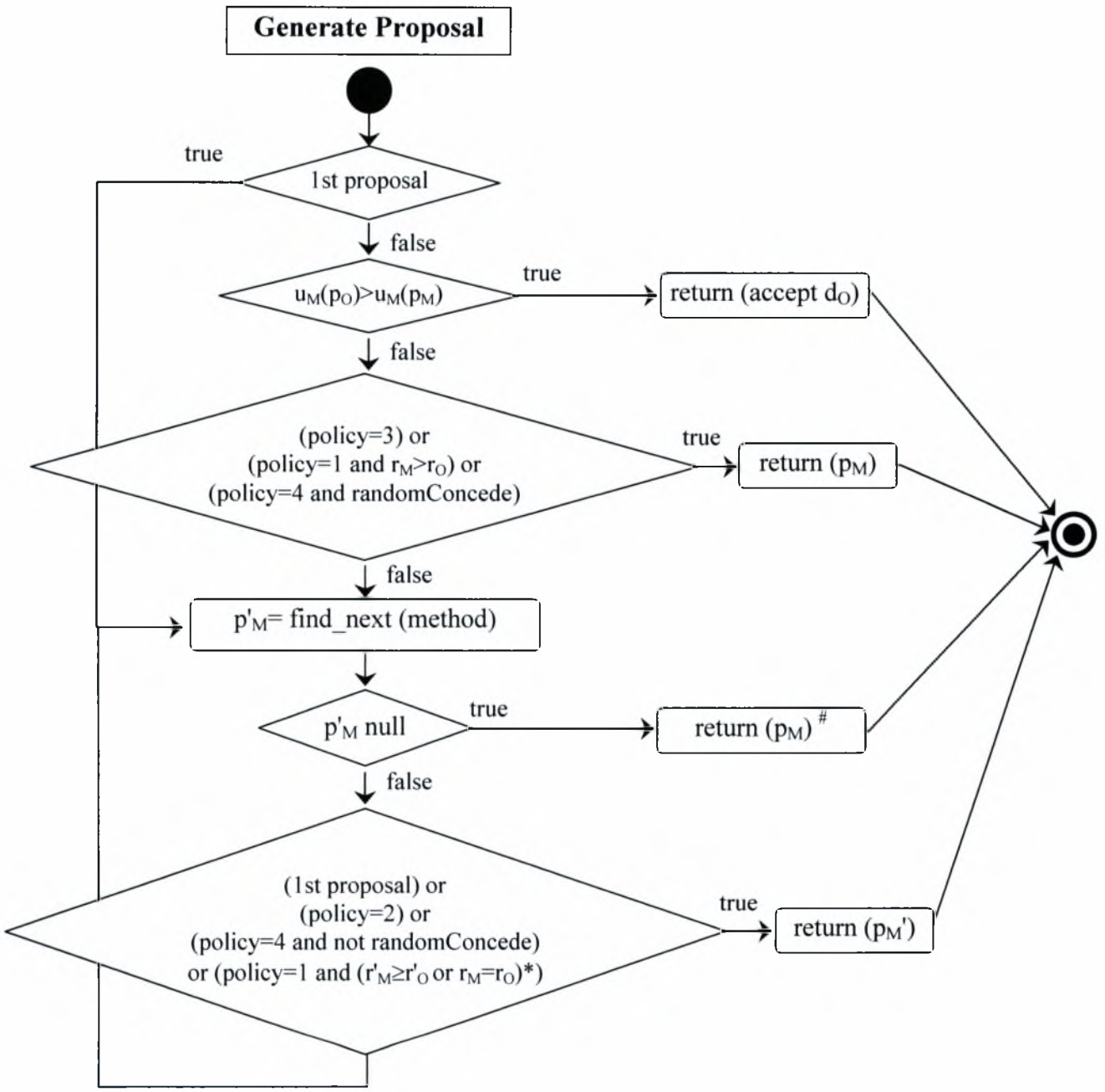
Διάγραμμα Π.2: Δραστηριότητα «Αποστολή πρότασης».



Διάγραμμα Π.3: Δραστηριότητα «Παραλαβή πρότασης».



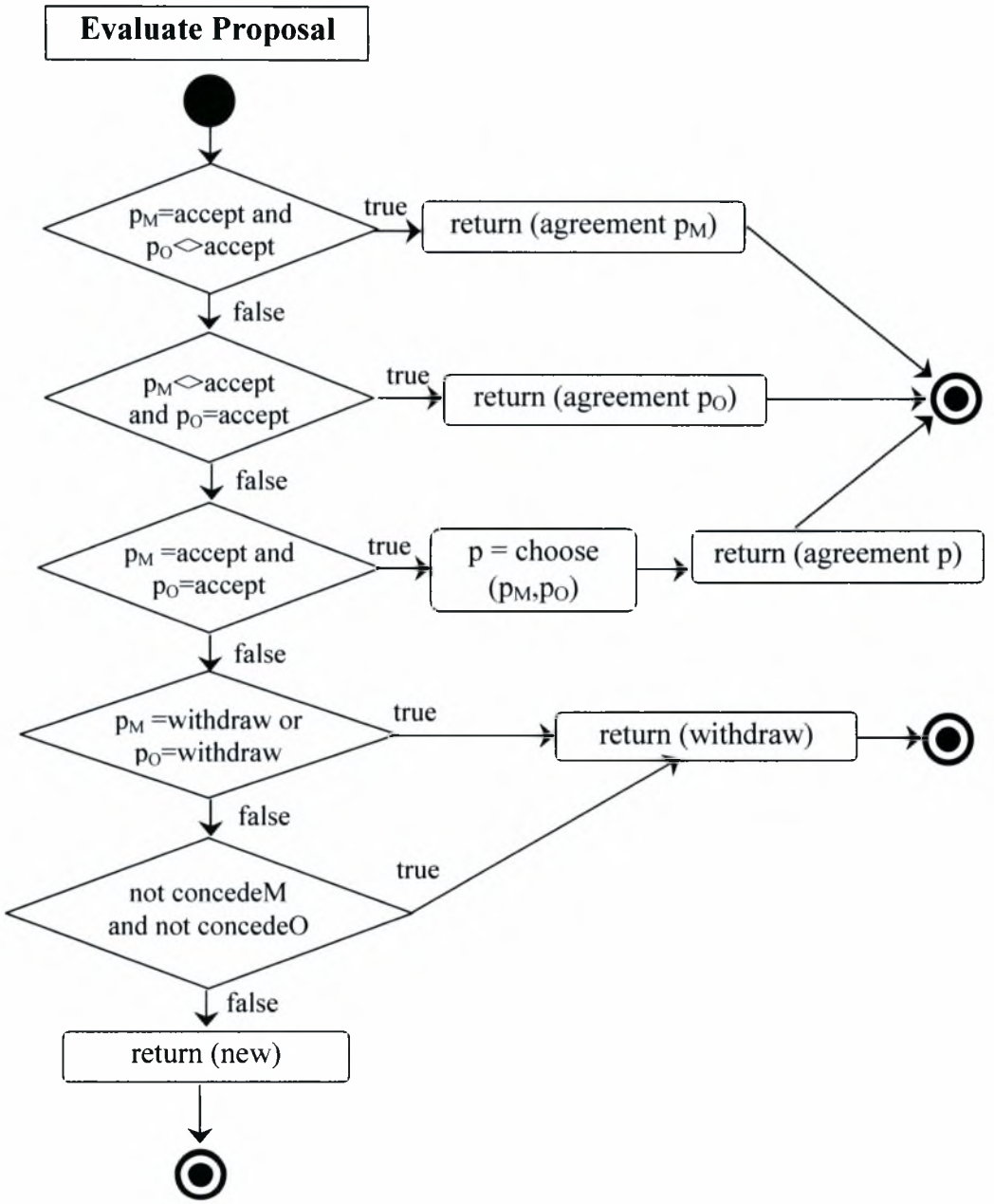
Διάγραμμα Π.4: Δραστηριότητα «Παραγωγή πρότασης».



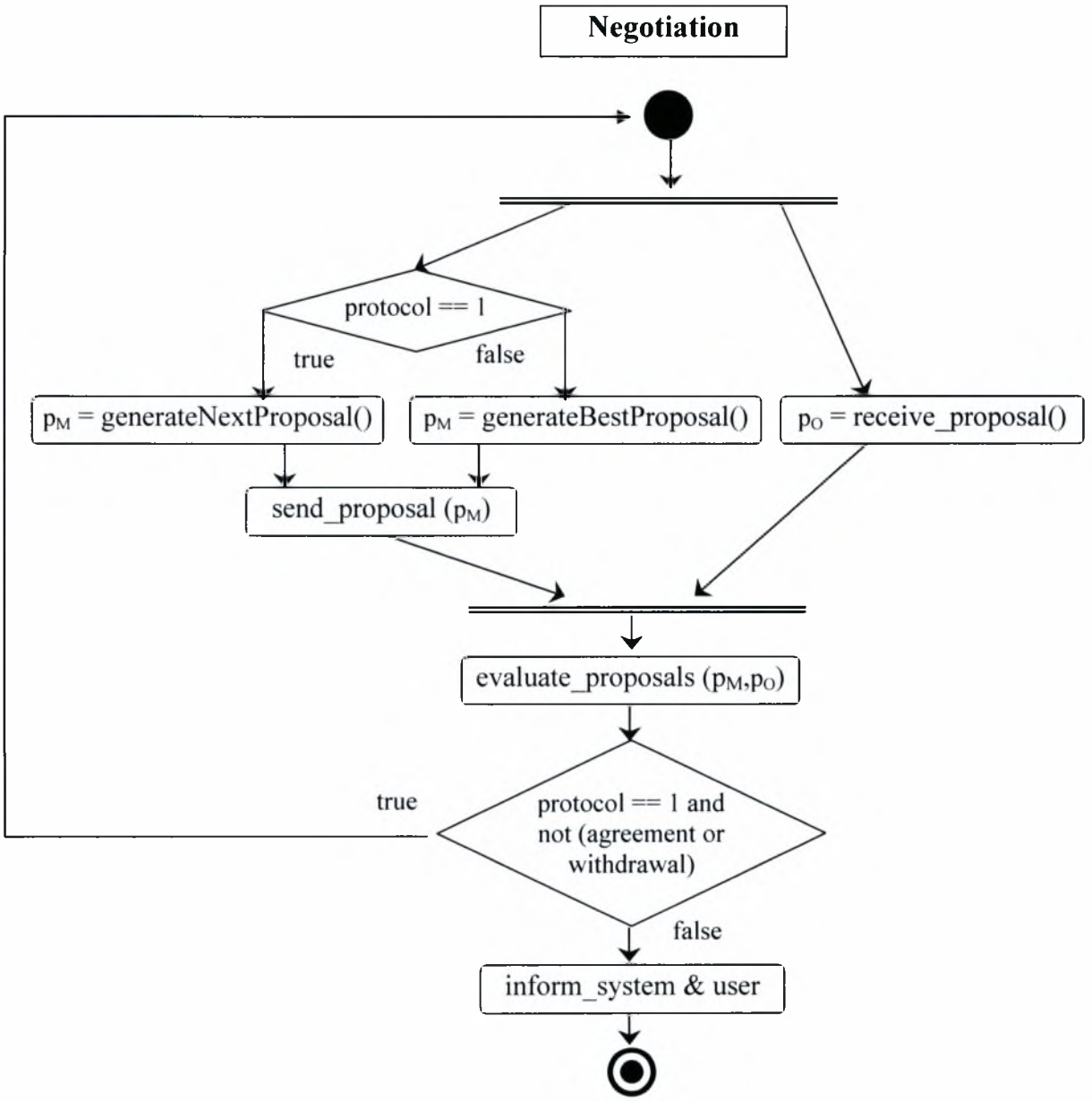
Στην περίπτωση ο πράκτορας δεν έχει νέα πρόταση να κάνει, προτείνει και πάλι την πρόταση του προηγούμενου γύρου.

* Συνθήκη που εξασφαλίζει την ελάχιστη ικανή υποχώρηση. Η ανισότητα $r'_M \geq r'_O$ αναφέρεται στο νέο γύρο. Η ισότητα $r_M=r_O$ αναφέρεται στον προηγούμενο γύρο και υποδεικνύει ότι και οι δυο πράκτορες υποχώρησαν.

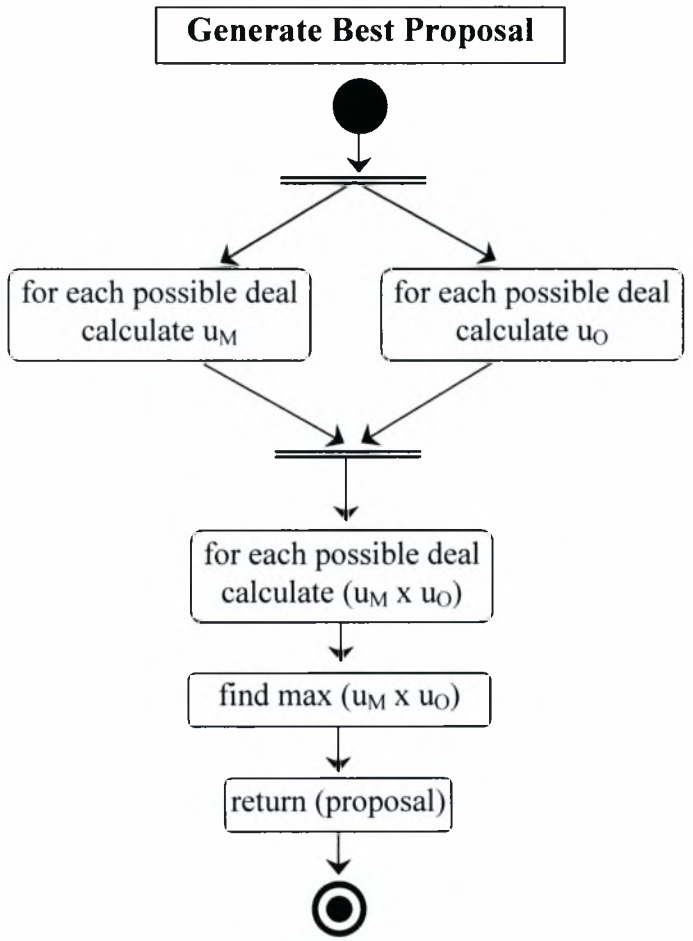
Διάγραμμα Π.5: Δραστηριότητα «Αποτίμηση πρότασης».



Διάγραμμα Π.1.1: Δραστηριότητα «Διαπραγμάτευση».
Αναφέρεται σε ΜΠΣ και ΠΕΒ. Αντικαθιστά το Π.1.



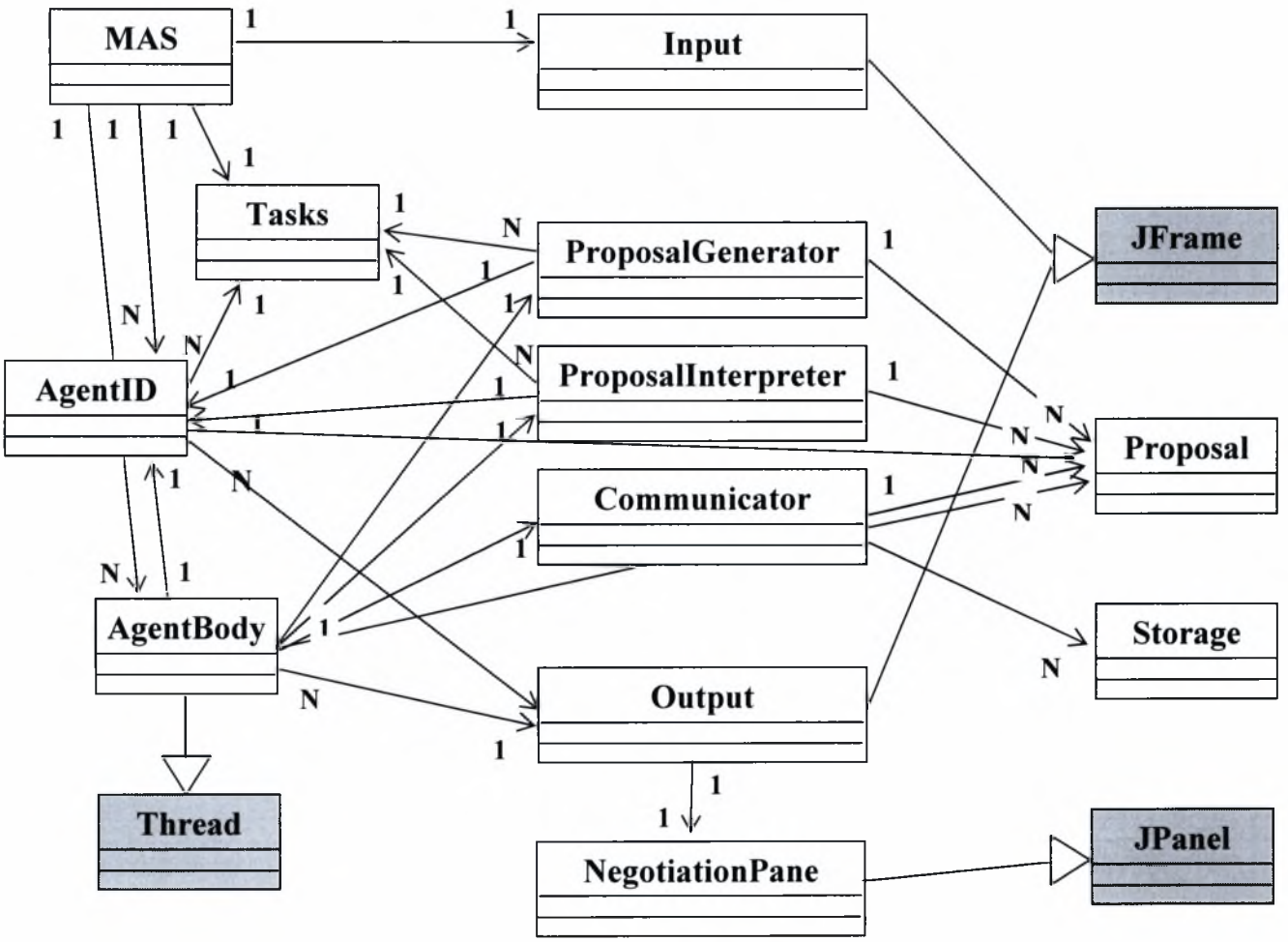
Διάγραμμα Π.6: Δραστηριότητα «Παραγωγή καλύτερης πρότασης».



Διαγράμματα και περιγραφή κλάσεων του συστήματος.

Διάγραμμα Π.7: Διάγραμμα κλάσεων.

Το παρόν διάγραμμα κλάσεων περιλαμβάνει μόνο τα ονόματα των κλάσεων του συστήματος, για να είναι πιο ευανάγνωστο. Η περιγραφή, τα χαρακτηριστικά και οι μέθοδοι των κλάσεων ακολουθούν. Σημειώνεται ότι οι JFrame, JPanel και Thread είναι κλάσεις της Java.



```
class Tasks {
    boolean complex; // true for complex, false for simple tasks
    int N; // number of simple tasks or created subtasks
    int Q; // number of complex tasks
    String[] name; // task names
    int[] number; // number of subtasks (complex tasks only)
    int[] cost; // cost of simple tasks & subtasks
    int[][] costI; // cost of complex tasks
    int totalCost; // total cost of all tasks
    int[] array2e; // array2e[i] = 2**i
    int[] dependency; // dependencies of subtasks (complex tasks only)
Tasks (boolean complex1, String[] name1, int[] a, int[][] b) {}
boolean getComplex() {} // returns true if tasks as complex
int getN () {} // returns N
int getQ () {} // returns Q
String getName (int i) {} // returns name of task i
int getNumber (int i) {} // returns number of subtasks for task i
int get2e (int i) {} // returns 2**i
boolean taskIsInSubset(int i, int subset) {}
                        // returns true if task or subtask i exists in a subset of tasks
int getTotalCost() {} // returns total cost of all tasks
int getCost (int i) {} // returns cost of simple task or subtask i
int getCost (boolean[] dRow) {} // returns cost of a deal for an agent. dRow = d[agent]
int getCost (int[] ieRow) {}
                        // returns cost of a encounter for an agent. ieRow = ie[agent]
boolean isMaster (int i) {} // returns true, if subtask i is master
boolean isSlave (int j) {} // returns true, if subtask j is slave
boolean isSlave (int j, int i) {} // returns true, if subtask j is slave of i
int getMaster (int j) {} // returns master of subtask j
boolean hasMasterInDeal (int j, boolean dpart[]) {}
                        // returns true, if master of subtask j exists in deal
int getSlaveFromDeal (int i, boolean dpart[]) {}
                        // returns the first slave of subtask i it finds in deal
}
```

```
class Proposal {
    String type; // proposal type: offer, counteroffer, accept, withdraw
    int sender; // sender code number
    int receiver; // receiver code number
    boolean[][] deal; // in deal[0], true points to simple tasks or subtasks of agent 0
                        // in deal[1], true points to simple tasks or subtasks of agent 1
    Proposal (String type1, int sender1, int receiver1, boolean[][] deal1) {}
    boolean typeEquals (String s) {} // returns true if type is equal to s
    String getType () {} // returns type
    int getSender () {} // returns sender
    int getReceiver () {} // returns receiver
    boolean[][] getDeal () {} // returns deal
}
```

```
class Storage {
    String message;
    private boolean newMessage = false;
                                    // flag that becomes true when there is a new message
synchronized void sendMessage (String m) {}
                                   // sends a message, if there is empty space, and notifies
synchronized String receiveMessage() {}
                                          // receives a message, if there is one, and notifies
}
```

```
class Communicator {
    int myself; // code number for the agent
    Storage sIn, sOut; // 2 storages for sending and receiving messages
Communicator (int myself1, Storage s0, Storage s1) {}
void send (Proposal p) {} // convert proposal to String and send
Proposal receive () {} // receive String and convert to proposal
String proposalToString (Proposal p) {} // convert proposal to String
String dealToString (boolean[][] d) {} // convert deal to String
Proposal stringToProposal (String s) {} // convert String to proposal
}
```

```
class AgentBody extends Thread {
    AgentID aID;
    int myself, opponent; // code numbers for both agents
    Output out; // user interface window
    Communicator com; // responsible for communicating functions
    ProposalGenerator pg; // responsible for generating proposals
    ProposalInterpreter pi; // responsible for interpreting proposals
AgentBody (AgentID aID1, Storage sIn, Storage sOut) {}
public void run () {} // agent operation
}
```

```
class ProposalInterpreter {
    AgentID aID;
    int myself, opponent; // code number for both agents
    Tasks t;
    int uNewM, uNewO, uOldM, uOldO; // for Zeuthen strategy
    boolean concedeM, concedeO;
ProposalInterpreter (AgentID aID1) {}
String evaluate (Proposal pM, Proposal pO) {} // evaluates agent's proposal pM and
  // opponent's proposal pO and returns keywords: withdraw, accept pM, accept pO, new
}
```

```
class AgentID {
    int myself, opponent; // code numbers for both agents
    String[] name; // names for both agents
    int protocol, method; // code protocol (1,2) and method (1,2,3,4)
    double epsilon; // used in method 3
    int policy;  // 1:zeuthen, 2:alwaysConcede, 3:neverConcede, 4:randomConcede
    int randConcede;  // (%) probability for agent concession in case of policy 4
    int randWinner;  // last step criterion, if both agents accept opponent's proposal
    int delay; // time delay between rounds in msec
    Tasks t;
    int[][] ie; // initial encounter
    int icM, icO, totalCost; // cost of initial encounter for both agents, cost of all tasks
    Output out; // for user interface
    double riskCompare; // used to compare risks at every round (for simple tasks)
AgentID (int myself1, String[] name1, int protocol1, int method1, int policy1,
    int randConcede1, int randWinner1, Tasks t1, int[][] ie1, int delay1, Output out1) {}
int getMyself () {} // returns myself
String getName (int i) {} //returns the name of agent i
int getProtocol () {} // returns protocol
int getMethod () {} // returns method
double getDelta () {} // returns delta factor
int getPolicy () {} // returns policy
int getRandWinner () {} // returns randWinner
int getRandConcede () {} // returns randConcede
int getDelay () {} // returns delay
Tasks getT () {} // returns t of class Tasks
Output getOut () {} // returns user interface window
int getICM () {} // returns icM, cost of initial encounter for myself
int getICO () {} // returns icO, cost of initial encounter for opponent
int getTotalCost () {} // returns totalCost
int getTotal_ICO () {} // returns totalCost - icO
String getID () {} // returns agent ID: name+protocol+method+protocol
String getTString () {} // returns e.g. <Tname0,Tname1>
String getDString () {} // returns e.g. <Dname0,D name1>
String encounterToString () {} // converts initial encounter to String, for user interface
String dealToString(boolean[][] d) {} // converts deal to String, for user interface
String proposalToString (Proposal p) {} // converts proposal to String, for user interface
int getCost (boolean[][] d, int ag) {}
                                        // returns the cost of a deal for myself or the opponent
int getUtility (boolean[][] d, int ag) {}
        // returns the utility of a deal for myself or the opponent: initial cost - current cost
int getUtilityProd (boolean[][] d) {}
                        // returns the product of utilities of the two agents for the same deal
int getUtilityProdDif (boolean[][] d1, boolean[][] d2) {}
                                        // returns the difference of utility product for two deals
int getUtilitySum (boolean[][] d) {}
                            // returns the sum of utilities of the two agents for the same deal
int getUtilitySumDif (boolean[][] d1, boolean[][] d2) {}
                                            // returns the difference of utility sum for two deals
double getRisk (boolean[][] d1, boolean[][] d2, int ag) {}
                        // returns the agent risk calculated according to the Zeuthen strategy
double getRiskDif (boolean[][] d1, boolean[][] d2, int ag1, int ag2) {}
                                            // returns the difference of risk for two agents
}
```

```
class ProposalGenerator {
    AgentID aID;
    int myself, opponent; // code number for both agents
    Tasks t;
    int N; // N is the number of simple tasks or subtasks
    int e2N; // e2N = 2**N
    int[][] subsets, cM, cO; // used in methods 1 & 2
    int Nc = 0; // used in methods 1,3
    int W; // used in method 2
    int currentPos; // position of next possible deal
    int oldCostM, oldCostO; // cost of old deal for myself & opponent
    int[][] negotiationSet;
ProposalGenerator(AgentID aID1) {}
int[][]createNS () {} // for all deals d in negotiation set, finds couples
                      // (utility(d,myself),utility(d,opponent)) to be graphically represented
int[][] getNS() {} // return negotiationSet
Proposal generateNextProposal (Proposal pM, Proposal pO) {} // generates new
  // proposal pM if old proposals are pM and pO. if pM is null, it generates first proposal
void findAllDeals_m13s () {} // (for simple tasks) finds all deals (subsets & costs)
       // in negotiation set using method 1 or 3 and sorts them by cost at increasing order
int mergeSubsetsS1(int low, int high, int pivot) {}
int mergeSubsetsS3(int low, int high, int pivot) {}
boolean approximateFoundS(int working[][], int m, int oldM) {}
void findAllDeals_m13c () {} // (for complex tasks) finds all deals (subsets & costs)
       // in negotiation set using method 1 or 3 and sorts them by cost at increasing order
int mergeSubsetsC1(int low, int high, int pivot) {}
int mergeSubsetsC3(int low, int high, int pivot) {}
boolean approximateFoundC(int working[][], int m, int oldM) {}
boolean[][] findNextDeal_m13 () {}
                    // finds next possible deal in negotiation set created with method 1 or 3
void findAllDeals_m2s () {} // (for simple tasks) finds all deals (subsets & costs)
            // in negotiation set using method 2 and sorts them by cost at increasing order
boolean[][] findNextDeal_m2s () {} // (for simple tasks) finds next possible deal using 2
void findAllDeals_m2c () {} // (for complex tasks) finds all deals (subsets & costs)
             // in negotiation set using method 2 and sort them by cost at increasing order
boolean[][] findNextDeal_m2c () {}
                               // (for complex tasks) find next possible deal using method 2
boolean[][] findNextDeal_m4s (boolean[][] d) {}
                                 // (for simple tasks) find next possible deal using method 4
boolean[][] findNextDeal_m4c (boolean[][] d) {}
                               // (for complex tasks) find next possible deal using method 4
int calculateSubtaskCost(int i, int subset) {} // (for complex tasks) returns cost'(i), such
   // that,if I undertake subtask i,then costs will become costM+cost(i) and costO-cost'(i)
int calculateSubtaskCost(int i, boolean d[][]) {}
                                     // (for complex tasks, meth.4) returns cost'(i) such that,
       // if I undertake subtask i, then costs will become costM+cost(i) and costO-cost'(i)
boolean isValidChoice(int i, int subset) {}
        // (complex tasks) returns true if subtask i is master,or is slave and has it's master
      // in subset (used for finding a subtask of opponent that the agent could undertake)
boolean isValidChoice(int i, boolean[][] d) {} // (for complex tasks, method 4)
```

```
                    // returns true if subtask i is master,or is slave and has it's master in deal
boolean[][] subsetToDeal(int subset) {} // calculates d from subset
boolean[][] getComplementary(boolean[][] d) {} // calculates dO from dM
boolean[][] getComplementary(boolean[][] d, int index) {
                                        // calculates dO (d[opponent]) from dM for method 4
Proposal generateBestProposal() {} // generates proposal with one step protocol
boolean[][] findBestDeal_m13 () {} // find best deal that maximises
                                // the product of utilities for the two agents using method 1 or 3
boolean[][] findBestDeal_m2 () {} // find best deal that maximises
                                    // the product of utilities for the two agents using method 2
boolean[][] findBestDeal_m4 () {} // find best deal that maximises
                                    // the product of utilities for the two agents using method 4
}
```

```
class Input extends JFrame implements ActionListener {
    JPanel inputPane;
    JTextField an0, an1; // agent names text fields
    String[] agentName; // agent names
    JComboBox examples; // predefined examples combo box
    JComboBox pr; // negotiation protocol combo boxes
    int protocol; // negotiation protocol
    JComboBox m0, m1; // agent methods combo boxes
    int method0, method1; // agent methods
    JLabel e0L, e1L;  // ε approximation factor labels for method 3
    JTextField e0, e1; // ε approximation factor text fields for method 3
    double epsilon0 = 0, epsilon1 = 0; // ε approximation factor for method 3
    JComboBox p0, p1; // agent policies combo boxes
    int policy0, policy1; // agent policies
    JTextField rc0, rc1; // percent for conceding randomly text fields
    JLabel rc0L, rc1L, tnL, tnuL, tcL, eL;
    int randConcede0, randConcede1; // percent for conceding randomly
    JTextField tn; // task names text fields
    String[] taskName; // task names
    JTextField tnu; // tasks numbers text fields
    int[] taskNumber; // tasks number
    JTextField tc; // tasks cost text field for simple tasks
    JTextField tc0, tc1; // tasks cost text fields for complex tasks
    int[] taskCost; // tasks cost for simple tasks
    int[][] taskCostC; // tasks cost for complex tasks
    JTextField e0, e1; // initial encounter text fields
    int[][] encounter; // initial encounter
    JComboBox comp; // complex or simple combo box
    boolean complex; // complex or simple
    JTextField d; // delay text field
    int delay; // delay in msec
    JButton okB; // OK button
    boolean ok = false;
    // predefined Strings for protocols, methods, policies
    String prS1 = "1 MCP";
```

```
    String prS2 = "2 one-step";
    String mS1 = "1 offline calculation";
    String mS2 = "2 offline with DP";
    String mS3 = "3 trim lists by ε";
    String mS4 = "4 heuristic method";
    String pS1 = "1 zeuthen strategy";
    String pS2 = "2 always concede";
    String pS3 = "3 never concede";
    String pS4 = "4 random concede";
    // predefined Strings for examples
    String ex1 = "1 {t0,t1,t2,t3} {1,3,5,10} {t0,t1,t2,t3}{t1,t2,t3}";
    String ex11 = "1.1 {t0,t1,t2,t3} {1,3,5,10} {t0,t1,t2,t3}{t2,t3}";
    String ex12 = "1.2 {t0,t1,t2,t3} {1,3,5,20} {t1,t2,t3}{t0,t1,t3}";
    String ex13 = "1.3 {t0,t1,t2,t3} {1,3,5,10} {t0,t1,t2,t3}{t2,t3} as complex";
    String ex2 = "2 {f0(2)f1(1)f2(1)} {1+n,1+n,1+n}
                                        {f0(2)f1(1)f2(1)}{f0(2)f1(1)f2(1)}";
    String ex3 = "3 {t0,t1,t2,t3} {1,2,49,69} {t0,t1,t2,t3}{t1,t3}";
    String ex31 = "3.1 {t0,t1,t2,t3} {1,2,49,69} {t1,t2,t3}{t0,t1,t2}";
    String ex4 = "4 {f0(3)f1(1)} {1+n,1+n} {f0(3)f1(1)}{f0(3)f1(1)}";
    String ex5 = "5 {f0(3)f1(2)f2(1)} {2+n,2+2n,5+3n}
                                        {f0(2)f1(2)f2(1)} {f0(3)f1(2)f2(1)}";
    String ex51 = "5.1 {f0(3)f1(2)f2(1)} {2+n,2+2n,5+3n}
                                        {f0(2)f1(1)f2(1)}{f0(2)f1(1)}";
    String ex6 = "6 {t0,t1,f2(2)} {3,4,2+2n} {t0,t1,f2(1)}{f2(2)}";
    String ex7 = "7 {t0,t1,t2} {9,5,1} {t0,t1,t2}{t0}";
    String ex8 = "8 {t0,t1,t2,t3} {1,2,3,4} {t0,t1,t2,t3}{t2,t3}";
Input () {}
public void actionPerformed (ActionEvent evt) {} // responds to user actions
void useExamples (String s) {} // uses predefined examples
boolean getOk() {} // returns OK
String[] getAgentName() {} //returns agentName
int getProtocol0() {}  // returns protocol for agent0
int getProtocol1() {} // returns protocol for agent1
int getMethod0() {} // returns method for agent0
int getMethod1() {} // returns method for agent1
double getEpsilon0() {} // returns epsilon factor for agent0
double getEpsilon1() {} // returns epsilon factor for agent1
int getPolicy0() {} // returns policy for agent0
int getPolicy1() {} // returns policy for agent1
int getRandConcede0() {} // returns randConcede for agent0
int getRandConcede1() {} // returns randConcede for agent1
boolean getComplex () {} // returns complex
String[] getTaskName() {} // returns taskName
int[] getTaskNumber() {} // returns taskNumber
int[] getTaskCost() {} // returns taskCost for simple tasks
int[][] getTaskCostC() {} // returns taskCost for complex tasks
int[][] getEncounter() {} // returns initial encounter
int getDelay() {} // returns delay
}
```

```
class Output extends JFrame {
    JPanel pane; // main panel
    JPanel pane0, pane1; // panels for both agents
    JScrollPane scroll0, scroll1; // scroll panels for both agents
    NegotiationPane nPane; // panel for graphical representation
    int x0 = 200, y0 = 120; // dimensions for graphical representation panel
Output () {}
void print (String s, Color col, int myself) {} // print String s to the appropriate panel
void drawNS(int[][] negotiationSpace, int f) {}
                                        // draw points representing the negotiation set
void drawProposal(int u0, int u1, int agent) {} // draw point representing a proposal
void drawAgreement(int u0, int u1) {} // draw point representing an agreement
}
```

```
class NegotiationPane extends JPanel {
    Graphics2D comp2D;
    int[][] negotiationSet;
      // array with couples (utility(p,myself),utility(p,opponent))for each proposal in NS
    int[][] proposals0; // array with couples
                        // (utility(p,myself),utility(p,opponent))for each proposal by agent0
    int[][] proposals1; // array with couples
                        // (utility(p,myself),utility(p,opponent))for each proposal by agent1
    int proposals0Index, proposals1Index; // current proposal number
    int[] agreement; // couple (utility(p,myself),utility(p,opponent))
                                                        // for agreement proposal
    int x0, y0; // dimensions for graphical representation panel
    int factor = 5; // used to scale points
public NegotiationPane(int x1, int y1) {}
public void paint(Graphics comp) {}
 // draws coordinates, negotiation set points, proposal by each agent, agreement proposal
int convertX(int x) {} // scale & move x dimension
int convertY(int y) {} // scale & move y dimension
void setNS(int[][] negotiationSet1, int f) {} // determine negotiation set
void setProposal0(int u0, int u1) {} // add proposal for agent 0
void setProposal1(int u0, int u1) {} // add proposal for agent 1
void setAgreement(int u0, int u1) {} // determine agreement
}
```

```
public class MAS {
public static void main(String[] arguments) {
}
```