ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ

Σχολή Μηχανικών Ηλεκτρονικών Υπολογιστών & Πληροφορικής

Διπλωματική Εργασία

με τίτλο:

Προσεγγιστικές ισορροπίες Nash

Πειραματική και Θεωρητική εργασία για εύρεση αποδοτικών αλγορίθμων που να δίνουν καλές προσεγγίσεις σε Nash ισορροπίες σε παίγνια 2 παικτών.

Επιβλέπων Καθηγητής: *Dr. Παύλος Σπυράκης*

Συν-επιβλέπων
Ερευνητής - Στέλεχος ΕΑΙΤΥ: *Dr. Χαράλαμπος Τσακνάκης*

Ακαδημαϊκό έτος 2007-2008

Εξάμηνο: 10ο

Φοιτητής:
Κανούλας Δημήτριος Α.Μ. 3109

Πάτρα Ιούλιος 2008

Περίληψη

Ένας από τους τομείς με τον οποίο ασχολείται η επιστήμη των υπολογιστών και ιδιαίτερα οι θεωρητικοί της επιστήμης των υπολογιστών, είναι η Αλγοριθμική Θεωρία Παιγνίων.

Η έρευνα που γίνεται πάνω σε ανοικτά προβλήματα της θεωρία παιγνίων είναι μεγάλη σήμερα. Ένα από τα ανοικτά προβλήματα αυτής, προσπαθούμε να ερευνήσουμε στην παρούσα διπλωματική εργασία.
Αυτό το πρόβλημα είναι ο υπολογισμός ισορροπιών Nash και πιο συγκεκριμένα ο υπολογισμός προσεγγιστικών ισορροπιών Nash (ε-approximate Nash Equilibria).

Η παρούσα διπλωματική εργασία αποτελεί την υλοποίηση του πρόσφατου αλγορίθμου που περιγράφεται στην δημοσιευμένη εργασία του Χ. Τσακνάκη και του Π. Σπυράκη [1].

Στην παρούσα αναφορά θα γίνει περιγραφή της υλοποίησης (δηλαδή του πώς τροποποιήθηκε πρακτικά ο θεωρητικός αλγόριθμος) και της συμπεριφοράς του αλγορίθμου αυτού. Εκτός των θεμάτων αυτών, θα περιγραφθούν επίσης κάποια θεωρητικά στοιχεία της Θεωρίας Παιγνίων, καθώς και κάποια στοιχεία του Γραμμικού Προγραμματισμού που είναι βασικά για την κατανόηση της έρευνας αυτής.

Αφορμή της έρευνας αποτέλεσε η υποψία ότι η πειραματική συμπεριφορά του αλγορίθμου αυτού, είναι πολύ καλύτερη από την συμπεριφορά που παρουσιάστηκε μέσω της θεωρητικής ανάλυσης, τόσο σε απόδοση που αφορά τον αριθμό των βημάτων εκτέλεσης, όσο και στην τιμή της προσεγγιστικής ισορροπίας Nash που υπολογίζεται.

Μέσω της υλοποίησης αυτής βγαίνουν αρκετά σημαντικά συμπεράσματα που αφορούν όχι μόνο τον υπολογισμό των προσεγγιστικών ισορροπιών Nash και την πολυπλοκότητα υπολογισμού αυτών, αλλά και τον υπολογισμό οποιασδήποτε ακριβής ισορροπίας Nash (exact Nash Equilibrium). Αυτό, γιατί αποδεικνύεται ότι ο μέχρι στιγμής καλύτερος πολυωνυμικός αλγόριθμος για τον υπολογισμό προσεγγιστικών ισορροπιών Nash που μελετάμε, δίνει στην πράξη πολύ καλά αποτελέσματα τόσο όσο αφορά την απόδοση στην ταχύτητα σύγκλισης, όσο και στην ποιότητα των αποτελεσμάτων για μη-συνεργατικά διπαίγνια. Τα αποτελέσματα είναι πολύ καλύτερα από την θεωρητικά χειρότερη περίπτωση που αναφέρεται στην δημοσίευση [1].

Η διπλωματική εργασία αυτή, εκτός της υλοποίησης και της περιγραφής του αλγορίθμου που αναφέρεται στην δημοσίευση [1], περιέχει ένα πλήθος πειραμάτων πάνω σε διάφορα είδη παιγνίων. Μέσα σε αυτά περιέχονται παίγνια που θεωρούνται δύσκολα. Δηλαδή παίγνια που υπό κανονικές συνθήκες θα περιμέναμε να δώσουν μεγάλες προσεγγιστικές ισορροπίες Nash. Τέτοιου είδους παίγνια ορίστηκαν και υλοποιήθηκαν σε αυτήν την διπλωματική εργασία. Χρησιμοποιήθηκαν στα πειράματά μας και μέσω αυτών βγάλαμε πολύ σημαντικά και εντυπωσιακά αποτελέσματα που αποδεικνύουν περαιτέρω την πολύ καλή συμπεριφορά του αλγορίθμου και δίνουν αποτελέσματα που μπορούν να χρησιμοποιηθούν στο μέλλον για μελέτη της πολυπλοκότητας υπολογισμού ισορροπιών Nash και στην έρευνα πάνω σε πολλά ανοικτά προβλήματα της Αλγοριθμικής Θεωρίας Παιγνίων.

Λέξεις Κλειδιά
Προσεγγιστική ισορροπία Nash, πολυωνυμική προσέγγιση, βελτιστοποίηση

Keywords
Approximate Nash Equilibria, polynomial approximations, optimization

Περιεχόμενα

Ευχαριστίες

Αρχικά θα ήθελα να ευχαριστήσω θερμά τον καθηγητή μου Παύλο Σπυράκη, ο οποίος από την πρώτη στιγμή με εμπιστεύτηκε και μου έδωσε την πολύτιμη ευκαιρία να συνεργαστώ μαζί του. Θεωρώ ότι υπήρξε για μένα ο άνθρωπος που με την καθοδήγησή του και την εμπειρία του, μου έδειξε τους ορίζοντες της θεωρητικής πληροφορικής, τόσο πριν την εκπόνηση της διπλωματικής μου εργασίας, όσο και κατά την διάρκεια αυτής. Οι συντονισμένες υποδείξεις του και οι παροτρύνσεις κατά την διάρκεια της έρευνας αυτής υπήρξαν πολύτιμες και σημαντικές για μένα.

Επίσης θέλω να ευχαριστήσω θερμά τον ερευνητή Μπάμπη Τσακνάκη, χωρίς την βοήθεια του οποίου αυτή η διπλωματική δεν θα είχε πάρει αυτήν την μορφή που έχει σήμερα, ούτε θα είχε προχωρήσει σε τέτοιο μεγάλο βαθμό. Η συχνή επικοινωνία και η βοήθεια του τόσο σε θεωρητικής φύσεως θέματα και προβλήματα που αντιμετώπιζα, όσο και σε θέματα και προβλήματα υλοποίησης, ήταν κάτι παραπάνω από πολύτιμη. Τον ευχαριστώ για την κατανόηση και την υπομονή που έδειξε καθ` όλη την διάρκεια της έρευνας αυτής.

Τέλος θα ήθελα να ευχαριστήσω τόσο την οικογένεια μου, όσο και τους φίλους μου, και ιδιαίτερα τους Βασίλη Ανανιάδη, Χρήστο Αποστόλου, Δημήτρη Καραμπίνα, Ανδρέα Κουτσόπουλο και Κώστα Πλατανιώτη, που αλτρουιστικά προσφέρθηκαν όλα αυτά τα χρόνια να με βοηθήσουν όποτε παρουσιάζονταν κάποιο πρόβλημα στην έρευνα μου.

Κεφάλαιο 1

Εισαγωγή

Ιστορικά η Θεωρία Παιγνίων (Game Theory) ξεκίνησε ως κλάδος και πεδίο των οικονομικών. Η αρχική ανάπτυξη της, αποδίδεται στους John von Neumann και Oskar Morgenstern οι οποίοι με το βιβλίο τους: "Theory of Games and Economic Behaviour" (Θεωρία Παιγνίων και Οικονομική Συμπεριφορά) πάνω σε παιχνίδια μηδενικού αθροίσματος (zero-sum games), μελέτησαν και όρισαν τη σχέση της θεωρίας παιγνίων με τον γραμμικό προγραμματισμό.
Αργότερα ο George B. Dantzig ανέπτυξε τον αλγόριθμο Simplex του γραμμικού προγραμματισμού μέσω του οποίου μπόρεσαν να επιλυθούν πολλά προβλήματα της θεωρίας παιγνίων.
Η μεγάλη τομή στην Θεωρία Παιγνίων έγινε από τον αμερικάνο μαθηματικό John Nash, μέρος της εργασίας του οποίου αποτελεί και ο ορισμός της ισορροπίας Nash, που θα ορίσουμε σε επόμενο κεφάλαιο. Για την εργασία του αυτή τιμήθηκε με το βραβείο Nobel οικονομίας.
Η Θεωρία Παιγνίων εφαρμόζεται σε πολλούς κλάδους των οικονομικών όπως στη βιομηχανική οργάνωση, στο σχεδιασμό μηχανισμών καις την Πολιτική Οικονομία.

Η Θεωρία Παιγνίων και η έννοια της ισορροπίας Nash όμως, πέρα από την οικονομική επιστήμη σύντομα άρχισαν να εφαρμόζονται και σε άλλα επιστημονικά πεδία. Πλέον η Θεωρία Παιγνίων αποτελεί έναν βασικό τομέα της επιστήμης υπολογιστών και χρησιμοποιείται σε πολλές εφαρμογές. Ο σπουδαιότερος είναι αυτός της μελέτης των δικτύων. Έτσι οι θεωρητικοί της πληροφορικής άρχισαν να ασχολούνται επίσημα πλέον με κεντρικά προβλήματα της θεωρίας παιγνίων, όπως είναι αυτός της ύπαρξης και του υπολογισμού της ισορροπίας Nash, με την οποία θα ασχοληθούμε και στην παρούσα εργασία. Πλήθος δημοσιεύσεων και μεγάλο μέρος της έρευνας των επιστημόνων της πληροφορικής στρέφεται πλέον στην επίλυση τέτοιων ή παρόμοιων προβλημάτων.

Μεγάλο μέρος της έρευνας κατευθύνθηκε στον υπολογισμό ισορροπίας Nash σε παίγνια με μικτές στρατηγικές, στα οποία οι παίκτες δεν επιλέγουν μία από τις στρατηγικές τους, αλλά παίζουν σύμφωνα με μια πιθανοτική κατανομή πάνω στο σύνολο των στρατηγικών τους. Από το θεώρημα του Nash γνωρίζουμε ότι κάθε παίγνιο έχει μικτή ισορροπία Nash. Όμως παραμένει ανοικτό το πρόβλημα του αν μπορεί αυτήν να υπολογιστεί σε πολυωνυμικό χρόνο.
Γνωρίζουμε ότι το πρόβλημα του υπολογισμού της ισορροπίας Nash ανήκει στην κλάση PPAD. Επίσης πρόσφατα αποδείχθηκε ότι το πρόβλημα είναι πλήρες στην κλάση αυτήν. Αλλά δεν ξέρουμε ακόμα αν μπορεί να υπολογιστεί σε πολυωνυμικό χρόνο.

Έτσι οι θεωρητικοί της πληροφορικής κατευθύνθηκαν στην έρευνα πάνω σε ένα παρόμοιο πρόβλημα. Προσπάθησαν να υπολογίσουν τις κατά προσέγγιση ισορροπίες Nash. Μέσω των προσεγγιστικών ισορροπιών Nash γίνεται μια προσπάθεια να υπολογιστεί η γνήσια ισορροπία Nash όπως ορίζει το θεώρημα Nash.
Μια τέτοια έρευνα γίνεται στην παρούσα εργασία. Μέσω της υλοποίησης της πρόσφατης δημοσίευσης των Χ. Τσακνάκη και Π. Σπυράκη [1], προσπαθήσαμε να δούμε στην πράξη τον υπολογισμό προσεγγιστικών ισορροπιών Nash μέσω του

αλγορίθμου που προτείνουν, ώστε να βγάλουμε κάποια συμπεράσματα που θα βοηθήσουν στην επίλυση του μεγάλου προβλήματος του υπολογισμού ισορροπιών Nash (exact Nash Equilibria).

Στο Κεφάλαιο 2 παρουσιάζουμε κάποιους θεωρητικούς ορισμούς που αναφέρονται στην Θεωρία Παιγνίων και σε όρους που θα χρειαστούμε στην συνέχεια και έχουν να κάνουν με τον υπολογισμό των ισορροπιών Nash.
Στο Κεφάλαιο 3 παρουσιάζουμε κάποιους ορισμούς που αναφέρονται στον γραμμικό προγραμματισμό, ο οποίος χρησιμοποιείται για να βρεθούν οι ισορροπίες Nash που αναζητούμε.

Το Κεφάλαιο 4 αποτελεί μια ιστορική αναδρομή και αναφορά στους αλγορίθμους που έχουν παρουσιαστεί μέχρι στιγμής και μελετούν το ίδιο πρόβλημα που κι εμείς ερευνούμε. Θα γίνει περιληπτική αναφορά στους αλγορίθμους και στις πολυπλοκότητες που αυτοί παρουσιάζουν.

Το Κεφάλαιο 5 είναι ένα από τα δύο σημαντικότερα κεφάλαια της εργασία αυτής. Παρουσιάζεται αναλυτικά ο αλγόριθμος τον οποίο υλοποιήσαμε, καθώς και όλες οι λεπτομέρειες της υλοποίησης αυτής. Το κίνητρο της υλοποίησης του αλγορίθμου που παρουσιάζεται στην εργασία του Χ. Τσακνάκη και Π. Σπυράκη [1], ήταν η υποψία ότι πρακτικά ο αλγόριθμος παρουσιάζει πολύ καλύτερη συμπεριφορά απ` ότι είχε υπολογιστεί θεωρητικά. Κάτι το οποίο τελικώς όπως θα δούμε όχι μόνο ισχύει, αλλά ξεπερνάει και τις αρχικές προσδοκίες μας.

Το Κεφάλαιο 6, ως τελευταίο κεφάλαιο παρουσιάζει όλα τα πειράματα που έγιναν για διάφορα προβλήματα και τα τελικά συμπεράσματα. Αποτελεί το σημαντικότερο κεφάλαιο μιας και εκεί τελικώς αποδεικνύεται η σπουδαιότητα της έρευνας που κάναμε στην παρούσα διπλωματική εργασία.

Κεφάλαιο 2

Παίγνια και Ισορροπία Nash

Στο κεφάλαιο αυτό παρουσιάζονται μερικές βασικές έννοιες της Θεωρίας Παιγνίων και της Αλγοριθμικής Θεωρία Παιγνίων.
Θα δοθεί έμφαση σε έννοιες που θα χρησιμοποιηθούν στην συνέχεια και αναφέρονται στον αλγόριθμο που υλοποιήσαμε.

2.1 Θεωρία Παιγνίων και Αλγοριθμική Θεωρία Παιγνίων

Αν θέλαμε να ορίσουμε τη Θεωρία Παιγνίων θα λέγαμε ότι είναι η μελέτη μοντέλων ανταγωνισμού και συνεργασίας μεταξύ ευφυών και λογικών οντοτήτων, που ονομάζονται παίκτες και δύναται να λαμβάνουν κάποιες αποφάσεις.
Έτσι η Θεωρία Παιγνίων είναι το πεδίο εκείνο που παρέχει μαθηματικές τεχνικές για την ανάλυση καταστάσεων όπου δύο ή περισσότερες οντότητες (παίκτες) καλούνται να πάρουν αποφάσεις που αλληλοεπηρεάζουν το όφελός τους.
Κάθε παίκτης παρουσιάζει την εγωιστική συμπεριφορά που έχει να κάνει με μεγιστοποίηση του κέρδους του. Αυτό αποτελεί την κυρίαρχη υπόθεση της ανάλυσης των διάφορων καταστάσεων και περιπτώσεων στα παίγνια.
Η Θεωρία Παιγνίων, όπως θα δούμε παρακάτω, ασχολείται με την ύπαρξη σημείων ισορροπίας μεταξύ των οντοτήτων αυτών, μεθόδους σύγκλισης προς αυτές τις ισορροπίες και την αναζήτηση ποιοτικών χαρακτηριστικών των οντοτήτων, όπως αυτά προκύπτουν λόγω της εγωιστικής τους συμπεριφοράς.

Η Αλγοριθμική Θεωρία Παιγνίων μελετά τα προβλήματα της Θεωρίας Παιγνίων από την οπτική γωνία της Θεωρητικής Πληροφορικής. Αυτό σημαίνει ότι για παράδειγμα εκτός του εντοπισμού κάποιας ισορροπίας στις επιλογές των παικτών που συμμετέχουν σε ένα παίγνιο, μας ενδιαφέρει ο εντοπισμός μιας τέτοιας ισορροπίας σε αποδοτικό χρόνο. Δηλαδή η σύγκληση σε μια τέτοια ισορροπία να γίνεται όσο τον δυνατόν ταχύτερα (σε πολυωνυμικό χρόνο για παράδειγμα). Επίσης ενδεχομένως να μας ενδιαφέρει ο εντοπισμός ισορροπιών που είναι περισσότερο επιθυμητές από άλλες, ανάλογα το σύστημα πάνω για τις οποίες τις υπολογίζουμε, καθώς και ο σχεδιασμός αποδοτικών αλγορίθμων και μηχανισμών για την εύρεση αυτών.

2.2 Παίκτες και Ωφέλεια Παικτών

Οι παίκτες στην Θεωρία Παιγνίων θεωρούνται λογικοί και ευφυείς.

Λογικός είναι ο παίκτης ο οποίος επιδιώκει να λάβει αποφάσεις που συμβάλουν στην επίτευξη των προσωπικών του στόχων. Όπως είπαμε η κυρίαρχη υπόθεση της Θεωρίας Παιγνίων είναι ότι οι παίκτες που συμμετέχουν σε ένα παίγνιο παρουσιάζουν μια εγωιστική συμπεριφορά. Δηλαδή στόχος κάθε παίχτη είναι πάντα η μεγιστοποίηση της προσωπικής αναμενόμενης τιμής του κέρδους του, το οποίο μετράται μέσω κάποιου ποσού ωφέλειας.
Στο βιβλίο τους οι Von Neumann και Morgenstern [3] διατύπωσαν, μέσω κάποιων παραδοχών, ότι υπάρχει τρόπος ανάθεσης ενός ποσού ωφέλειας σε κάθε ένα από τα διάφορα αποτελέσματα που θα έχει η επιλογή ενός παίκτη, σε συνδυασμό με τις επιλογές των άλλων παικτών.

Στην περίπτωσή μας, η ωφέλεια θα είναι ακέραιοι θετικοί αριθμοί. Όσο μεγαλύτεροι είναι αυτοί τόσο μεγαλύτερο είναι το κέρδος. Και κάθε παίκτης αποσκοπεί στο να αυξήσει την ωφέλεια του αυτή (Θεώρημα μεγιστοποίησης της αναμενόμενης ωφέλειας)
Οι τιμές αυτές (payoffs) για κάθε παίκτη, δίνονται βάσει μιας συνάρτησης απονομής κέρδους, η οποία ανάλογα με τις στρατηγικές που επιλέγει να παίξει ο κάθε παίκτης, αλλά και βάσει τις στρατηγικές που παίζουν οι άλλοι παίκτες αποδίδει μια τιμή κέρδους (payoff) στο παίκτη αυτόν. Η συνάρτηση απονομής κέρδους είναι ανεξάρτητη για κάθε παίκτη.

Ευφυής είναι ο παίκτης που έχει πλήρη γνώση της δομής και των κανόνων του παιγνίου που συμμετέχει και γνωρίζει τις στρατηγικές που μπορεί ο ίδιος, αλλά και οι άλλοι παίκτες να παίξουν. Επίσης γνωρίζοντας την κατάσταση στην οποία βρίσκεται το παίγνιο δύναται να εξάγει κάποια συμπεράσματα για το παίγνιο. Τέλος γνωρίζει ότι και οι άλλοι παίκτες είναι επίσης λογικοί και ευφυείς.

2.3 Μορφές και Είδη Παιγνίων

Υπάρχουν αρκετά μοντέλα περιγραφής ενός παιγνίου. Δύο είναι οι βασικές μορφές αναπαράστασης παιγνίων: η εκτατική (extensive) και η στρατηγική ή κανονική (normal) μορφή αναπαράστασης.

Από αυτές τις δύο θα περιγράψουμε μόνο την κανονική μορφή με την οποία θα αναπαριστούμε τα παίγνια στη συνέχεια της διπλωματικής εργασίας.
Παρ` όλα αυτά, ακόμα και σε εκτατική μορφή να δοθεί ένα παίγνιο υπάρχουν τρόποι μετατροπής του παιγνίου από εκτατική σε κανονική μορφή, όπως θα δούμε στην συνέχεια..

Στρατηγική ή Κανονική μορφή
Είναι ένας από τους απλούστερους τρόπους αναπαράστασης ενός παιγνίου.
Για να οριστεί το παίγνιο χρειάζεται να καθοριστούν:
- Το σύνολο των παικτών στο παίγνιο
- Το σύνολο των επιλογών που έχει κάθε παίκτης
- Τον τρόπο με τον οποίο εξαρτώνται τα κέρδη των παικτών από τις επιλογές τους

Έτσι λοιπόν μπορούμε να περιγράψουμε ένα παίγνιο Γ δίνοντας τα εξής στοιχεία:

$$\Gamma = (N, (C_i)_{i \in N}, (u_i)_{i \in N})$$

Όπου:
- Ν είναι ένα μη κενό σύνολο που εκφράζει το σύνολο των παικτών στο παίγνιο Γ

- C_i είναι ένα μη κενό σύνολο για κάθε παίκτη i που εκφράζει το σύνολο των αγνών στρατηγικών που έχει στη διάθεση του ο κάθε παίκτης i.
  Όπως θα δούμε και παρακάτω, στο παίγνιο Γ, κάθε παίκτης i καλείται να επιλέξει να παίξει μία από τις στρατηγικές αυτές (γνήσιες) ή ένα σύνολο από αυτές, βάσει μιας πιθανοτικής κατανομής πάνω στις γνήσιες στρατηγικές του.

Ορισμός 2.1 – Περίγραμμα Στρατηγικών

Ορίζουμε ως περίγραμμα στρατηγικών ένα συνδυασμό στρατηγικών που θα μπορούσαν να επιλέξουν οι παίκτες (ο καθένας ξεχωριστά) και συμβολίζουμε με C όλα τα πιθανά αυτά περιγράμματα στρατηγικών.

- Τότε για κάθε τέτοιο περίγραμμα στρατηγικών που ανήκει στο C, η $(u_i)_{i\in N}$ είναι μια συνάρτηση από αυτό το περίγραμμα, στο σύνολο των πραγματικών αριθμών και αναπαριστά το αναμενόμενο κέρδος για κάθε παίκτη i που παίζει το συγκεκριμένο περίγραμμα.

Οι παίκτες επιλέγουν ταυτόχρονα τις στρατηγικές τους, δηλαδή το περίγραμμα στρατηγικών δημιουργείται μη λαμβάνοντας υπόψη τον περιορισμό του χρόνου.

Επιπλέον αν για κάθε παίκτη $i\in N$ το σύνολο C_i είναι πεπερασμένο, τότε το παίγνιο ονομάζεται πεπερασμένο παίγνιο.

Παράδειγμα 2.1
Θα παρουσιάσουμε το παίγνιο που ονομάζεται «*Το δίλημμα του φυλακισμένου*»

Ιστορία:
Δύο άτομα έχουν συλληφθεί στον τόπο του εγκλήματος και ανακρίνονται.
Η μέγιστη ποινή είναι φυλάκιση 3 χρόνια, αλλά έχουν την ευκαιρία να κατηγορήσουν ο ένας τον άλλον ώστε να μειώσουν τα χρόνια φυλάκισής τους.

Κανόνες Παιγνίου:
Όποιος κατηγορήσει τον άλλο αφήνεται ελεύθερος [3 μονάδες ωφέλεια], αλλά ο άλλος πηγαίνει στην φυλακή για το μέγιστο χρόνο φυλάκισης [δηλαδή 0 μονάδες ωφέλεια].
Αν και οι δύο προδώσουν τότε πάνε και οι δύο στην φυλακή, αλλά για λιγότερα χρόνια γιατί συνεργάστηκαν με τις αρχές [1 μονάδα ωφέλεια]
Αν κανείς δεν προδώσει τότε πάνε φυλακή, αλλά η ωφέλεια τους είναι μεγαλύτερη από το να προδώσουν [2 μονάδες ωφέλεια].

Το παίγνιο παρουσιάζεται στον παρακάτω πίνακα:
Η προδοσία αντιστοιχεί στην στρατηγική D και η μη προδοσία στη στρατηγική C.
Οι παίκτες, έχουν να επιλέξουν την προδοσία (D) ή την μη-προδοσία (C).
Οι στρατηγικές του ενός παίκτη είναι στις γραμμές και του άλλου παίκτη στις στήλες.
Το σημείο του πίνακα που προκύπτει από τον συνδυασμό της στρατηγικής του παίκτη γραμμής και του παίκτη στήλης δίνει δύο αριθμούς. Ο πρώτος είναι το κέρδος του παίκτη γραμμής και ο άλλος το κέρδος του παίκτη στήλης.

	C	**D**
C	(2,2)	(0,3)
D	(3,0)	(1,1)

Περιπτώσεις Επιλογής Στρατηγικών:
Η καλύτερη στρατηγική για κάθε φυλακισμένο είναι να προδώσει τον άλλον, όταν ο άλλος δεν προδώσει (ωφέλεια 3 μονάδες).

Η επόμενη καλύτερη είναι όταν κανένας δεν προδώσει τον άλλον (ωφέλεια 2 μονάδες).
Η χειρότερη έκβαση θα ήταν για ένα φυλακισμένο όταν αυτός δεν προδώσει, ενώ ο άλλος προδώσει. Τότε έχει ωφέλεια 0 μονάδες.

-o-

Παρακάτω θα δούμε κάποια χαρακτηριστικά είδη παιγνίων που θα μας χρησιμέψουν στην συνέχεια της εργασίας, μιας και μερικά από αυτά θα τα χρησιμοποιήσουμε και στα πειράματά μας.

Παίγνια μηδενικού αθροίσματος (Zero-sum games)
[*Θα γίνει ορισμός για παίγνια zero-sum δύο παικτών*]

Ένα παίγνιο μεταξύ 2 παικτών ονομάζεται zero-sum όταν για μια συγκεκριμένη κατάσταση, δηλαδή για κάποιον συνδυασμό στρατηγικών, το κέρδος (payoff) του ενός ισούται με την απώλεια (loss)/κόστος(cost) του άλλου.

Σε ένα zero-sum παίγνιο το συνολικό κέρδος (payoff), δηλαδή το άθροισμα των τιμών του κέρδους, και των δύο παικτών είναι πάντα μηδέν.

Σημείωση: Από τον ορισμό προκύπτει ότι αν σε μια δεδομένη επιλογή στρατηγικών των δύο παικτών το κέρδος του ενός παίκτη είναι μηδέν τότε και του άλλου παίκτη το κέρδος είναι μηδέν. Όπως θα δούμε παρακάτω αυτό το σημείο είναι ένα σημείο που υπάρχει ισορροπία Nash.

Παράδειγμα 2.2
Παρακάτω παρουσιάζεται ένα παίγνιο μηδενικού αθροίσματος.
Οι γραμμές του πίνακα αντιστοιχούν στις στρατηγικές του πρώτου παίκτη και οι στήλες στις στρατηγικές του δεύτερου παίκτη.

Πίνακας Κερδών		
	y1	y2
x1	(4,-4)	(-2,2)
x2	(-3,3)	(0,0)

-o-

Win-loose Παίγνια
Είναι εκείνα τα παίγνια που το κέρδος κάθε παίκτη για οποιαδήποτε αγνή στρατηγική παιχτεί, είναι είτε 0 είτε 1.Με άλλα λόγια είτε κερδίζει, είτε χάνει (δεν υπάρχει ενδιάμεση κατάσταση κέρδους).

Παράδειγμα 2.3
Παρακάτω παρουσιάζεται ένα παίγνιο win-lose.
Οι γραμμές του πίνακα αντιστοιχούν στις στρατηγικές του πρώτου παίκτη και οι στήλες στις στρατηγικές του δεύτερου παίκτη.

Πίνακας Κερδών		
	y1	y2

x1	(0,1)	(1,0)
x2	(1,1)	(0,0)

-o-

2.4 Πίνακες Κέρδους (Payoff Matrices)

Στην παρούσα εργασία μελετούμε παίγνια που έχουν μόνο δύο παίκτες. Αυτά ονομάζονται διπαίγνια (Bimatrix Games). Από εδώ και πέρα δεν θα αναφερόμαστε σε ορισμούς που αφορούν γενικά παίγνια, αλλά σε διπαίγνια.

Οι δύο παίκτες, όπως θα καταλάβετε παρακάτω, ονομάζονται:
παίκτης γραμμής ο ένας και παίκτης στήλης ο άλλος.

Τα παίγνια που μελετούμε είναι μη-συνεργατικά. Δηλαδή οι παίκτες δεν συνεργάζονται μεταξύ τους κατά την διάρκεια του παιγνίου ώστε να μεγιστοποιήσουν το κέρδος τους. Τέτοιου είδους παίγνια χρησιμοποιήθηκαν κατά κόρον ώστε να κατανοηθούν φαινόμενα που έχουν να κάνουν με παίκτες που οι αποφάσεις τους είναι αλληλεπιδραστικές.

Τα κέρδη (payoffs) που έχει κάθε παίκτης περιγράφονται μέσω δύο πινάκων R, C.
Τα στοιχεία του πίνακα R περιέχουν τα κέρδη που λαμβάνει ο παίκτης 1 (παίκτης γραμμής) και τα στοιχεία του πίνακα C περιέχουν τα κέρδη που λαμβάνει ο παίκτης 2 (παίκτης στήλης).
Αν λοιπόν ο πρώτος παίκτης (παίκτης γραμμής) δύναται να παίξει m γνήσιες στρατηγικές και ο δεύτερος παίκτης (παίκτης γραμμής) n γνήσιες στρατηγικές τότε οι πίνακες R και C έχουν μέγεθος m x n.
Οι γραμμές των δύο πινάκων αντιστοιχούν στις στρατηγικές του παίκτη 1, για αυτό λέγεται και παίκτης γραμμής και οι στήλες αντιστοιχούν στις στρατηγικές του παίκτη 2, για αυτόν τον λόγο ονομάζεται παίκτης στήλης.
Κάθε στοιχείο του πίνακα R δίνει το όφελος του παίκτη γραμμής, αν ο παίκτης γραμμής επιλέξει να παίξει την στρατηγική που αντιστοιχεί στη γραμμή του εν λόγω στοιχείου και ο παίκτης στήλης επιλέξει να παίξει τη στρατηγική που αντιστοιχεί στη στήλη του εν λόγω στοιχείου.
Ομοίως ισχύει για τον πίνακα C που δίνει το όφελος του παίκτη στήλης αντίστοιχα.

Παράδειγμα 2.4
Στο παραπάνω παράδειγμα των φυλακισμένων οι αντίστοιχοι πίνακες κέρδους (R, C) που περιγράψαμε παραπάνω είναι οι εξής:

R		
	C	**D**
C	2	0
D	3	1

C		
	C	**D**
C	2	3

D	0	1

-o-

2.5 Στρατηγικές Παικτών

Παρακάτω θα γίνει ανάλυση που αφορά τις στρατηγικές που μπορούν να παίξουν οι παίκτες κατά την διάρκεια ενός παιγνίου.

Γνήσιες Στρατηγικές

Όπως είδαμε παραπάνω κάθε παίκτης έχει να επιλέξει μεταξύ κάποιων στρατηγικών. Στα διπαίγνια, έστω ότι ο παίκτης γραμμής μπορεί να επιλέξει μεταξύ m στρατηγικών και ο παίκτης στήλης μεταξύ n στρατηγικών (ανεξάρτητες από αυτές του παίκτη γραμμής). Αυτές θα λέμε ότι είναι οι γνήσιες στρατηγικές των δύο παικτών.

Μικτές Στρατηγικές

Όμως στα παραδείγματα που τρέχουμε οι παίκτες δεν καλούνται απαραίτητα να επιλέξουν να παίξουν μία από τις γνήσιες στρατηγικές τους, αλλά μπορούν να παίξουν ένα σύνολο στρατηγικών, δηλαδή κάποια μικτή στρατηγική.

Δοθέντος ενός παιγνίου σε στρατηγική μορφή $\Gamma = (N,(C_i)_{i\in N},(u_i)_{i\in N})$, μια μικτή στρατηγική για τον παίκτη i είναι μια πιθανοτική κατανομή πάνω στο C_i.

Στην περίπτωσή μας, την πιθανοτική αυτή κατανομή για τον παίκτη γραμμής θα την συμβολίσουμε με το διάνυσμα x μεγέθους m και για τον παίκτη στήλης θα την συμβολίσουμε με το διάνυσμα y μεγέθους n. το άθροισμα όλων των στοιχείων του x είναι ίσο με 1. Ομοίως για το άθροισμα όλων των στοιχείων του y.

Δηλαδή πιο φορμαλιστικά ισχύει:

$x = (x_1, x_2, ..., x_m)$, όπου $x_i \geq 0$ και $\sum x_i = 1$ για κάθε $0 \leq i \leq m$

και

$y = (y_1, y_2, ..., y_n)$, όπου $y_i \geq 0$ και $\sum y_i = 1$ για κάθε $0 \leq i \leq n$

Τα διανύσματα x και y καθορίζουν ένα περίγραμμα μικτών στρατηγικών. Υπάρχουν άπειρα τέτοια περιγράμματα μικτών στρατηγικών, ανάλογα με τις ανεξάρτητες επιλογές των πιθανοτικών κατανομών που καλούνται να επιλέξουν οι παίκτες.

Παράδειγμα 2.5:

Στο παραπάνω παράδειγμα των φυλακισμένων μία πιθανή μικτή στρατηγική που μπορεί να παίξει ο παίκτης γραμμής θα ήταν x=[0.5, 0.5], δηλαδή κάθε αγνή στρατηγική του με ίση πιθανότητα.

Ενώ ο παίκτης στήλης με μια οποιαδήποτε άλλη κατανομή: y=[0.7, 0.3], δηλαδή την πρώτη στρατηγική με πιθανότητα 70% και την δεύτερη με πιθανότητα 30%.

-o-

Αναμενόμενες τιμές κέρδους

Όταν ο παίκτης γραμμής επιλέγει να παίξει την μικτή στρατηγική x και ο παίκτης στήλης επιλέγει να παίξει μια μικτή στρατηγική y, τότε η αναμενόμενη τιμή κέρδους για τον παίκτη γραμμής ισούται με: $x^T Ry$

Επίσης η αναμενόμενη τιμή κέρδους για τον παίκτη στήλης ισούται με: $x^T Cy$.

Σημείωση: Θα αναφερόμαστε στις στρατηγικές που παίζουν δύο παίκτες ως το ζεύγος των δύο διανυσμάτων (x,y)

Support μιας μικτής στρατηγικής

Αν το $x_i \succ 0$ τότε λέμε ότι η μικτή στρατηγική x χρησιμοποιεί την i-στη αγνή στρατηγική. Το support του x (Supp(x)) είναι το σύνολο όλων εκείνων των αγνών στρατηγικών που χρησιμοποιούνται στην μικτή στρατηγική x (δηλαδή που έχουν θετική πιθανότητα)

Best response μιας μικτής στρατηγικής

Ως best response θα ορίσουμε το σύνολο εκείνων των στρατηγικών που μπορεί να παίξει ένας παίκτης και να του δώσει το βέλτιστο κέρδος, όταν οι στρατηγικές των άλλων παικτών είναι δεδομένες.

Παράδειγμα 2.6

Θεωρούμε ένα διπαίγνιο που ορίζεται από τον εξής πίνακα κέρδους των δύο παικτών:

Πίνακας Κερδών		
	y1	y2
x1	(4,-4)	(-2,2)
x2	(-3,3)	(0,0)

Αν ο παίκτης γραμμής επιλέξει να παίξει την στρατηγική x1, τότε το best response του παίκτη στήλης είναι η στρατηγική y2, γιατί σε αντίθεση με την y1 του επιφέρει το υψηλότερο κέρδος (Με την y1 στρατηγική θα είχε κέρδος -4, ενώ με την y2, που αποτελεί best response στην x1, έχει κέρδος +2)

-o-

Σημείωση: Πιστεύεται ότι οι παίκτες που συμμετέχουν σε ένα παίγνιο τείνουν στο να παίξουν όσο τον δυνατόν πιο απλές στρατηγικές. Μεταξύ άλλων αυτό δείχθηκε στα [22] και [23]. Φαίνεται να προτιμούν οι παίκτες να παίζουν στρατηγικές που ενδεχομένως να μην δίνουν την βέλτιστη λύση, αλλά είναι εύκολες στο να υλοποιηθούν. Απλές στρατηγικές είναι για παράδειγμα οι στρατηγικές που έχουν ομοιόμορφη κατανομή πάνω σε μικρά supports.

2.6 Ισορροπία Nash

Βασική έννοια την οποία ουσιαστικά μελετούμε στην παρούσα εργασία είναι η έννοια της ισορροπίας Nash.

Αναφερόμαστε πάντα σε παίγνια μη-συνεργατικά (όπου οι παίκτες δεν συνεργάζονται μεταξύ τους) και στο οποίο οι παίκτες είναι λογικοί και ευφυείς και λειτουργούν, όπως είδαμε, ανεξάρτητα όταν επιλέγουν τις στρατηγικές τους.
Ένα παίγνιο ισορροπεί όταν οι παίκτες επιλέγουν να παίξουν κάποια στρατηγική ο καθένας και κανένας παίκτης δεν έχει κίνητρο να αποκλίνει μονομερώς από τη στρατηγική του, δηλαδή κανείς παίχτης δεν μπορεί να αυξήσει το κέρδος του επιλέγοντας να παίξει μια διαφορετική στρατηγική όταν οι άλλοι παίκτες διατηρούν τις στρατηγικές τους.

Πιο φορμαλιστικά:

Ορισμός Ισορροπίας Nash (Nash Equilibrium)
Ένα ζεύγος στρατηγικών $(\tilde{x}, \tilde{y})$ είναι ισορροπία Nash για ένα διπαίγνιο, όταν ισχύει ότι:

1) Για κάθε πιθανή μικτή στρατηγική x του παίκτη γραμμής: $x^T R\tilde{y} \le \tilde{x}^T R\tilde{y}$ και
2) Για κάθε πιθανή μικτή στρατηγική y του παίκτη στήλης: $\tilde{x}^T Cy \le \tilde{x}^T C\tilde{y}$

Που σημαίνει στην περίπτωση (1) ότι αν x η πιθανοτική κατανομή στρατηγικών στην οποία μονομερώς αλλάζει ο παίκτης γραμμής, τότε το κέρδος που θα έχει αν παίξει αυτήν, όποια κι αν είναι αυτήν, θα είναι πάντα μικρότερο.
Ομοίως για την περίπτωση (2), για τον παίκτη στήλης.
Άρα δεν συμφέρει σε κανέναν από τους δύο παίκτες να αποκλίνει από την στρατηγική $(\tilde{x}, \tilde{y})$ αντίστοιχα.

Παράδειγμα 2.7
Θεωρούμε πάλι το παίγνιο των φυλακισμένων που περιγράφεται με τον εξής πίανακα:

	C	**D**
C	(2,2)	(0,3)
D	(3,0)	(1,1)

Στο παραπάνω παράδειγμα των φυλακισμένων βλέπουμε ότι υπάρχει μία αγνή ισορροπία Nash. Κι αυτή είναι να παίξουν και οι δύο παίκτες την στρατηγική D (δηλαδή να προδώσουν και οι δύο παίκτες). Ακόμα κι αν αυτό μοιάζει παράδοξο είναι η μόνη στρατηγική που κανέναν παίκτη δεν συμφέρει μονομερώς να παρεκκλίνει γιατί χάνει από το κέρδος του.
Αν παρεκκλίνει ο παίκτης γραμμής τότε από την στρατηγική (D,D) πηγαίνουμε στην στρατηγική (C,D) το κέρδος του παίκτη γραμμής γίνεται από 1 γίνεται 0. Ομοίως αν παρεκκλίνει ο παίκτης στήλης από την στρατηγική (D,D) πηγαίνουμε στην στρατηγική (D,C) το κέρδος του παίκτη γραμμής γίνεται από 1 γίνεται 0.
Άρα δεν συμφέρει κανέναν να παρεκκλίνει μονομερώς. Και άρα και οι δύο θα επιλέξουν την στρατηγική D.

-o-

2.7 Προσεγγιστική ή Κατά προσέγγιση Ισορροπία Nash

Παρ` ότι ο John Nash απέδειξε ότι υπάρχει πάντα μια ισορροπία Nash, η εύρεσή της ακόμα και για παίγνιο 2 παικτών σε πολυωνυμικό χρόνο είναι ένα πρόβλημα που

ακόμα δεν έχει λυθεί. Έτσι λόγω αυτού προέκυψε ο όρος της προσεγγιστικής ισορροπίας Nash (approximate Nash Equilibria) που είναι και γνωστή ως ε-Nash Equilibrium.
Ένα παίγνιο έχει προσεγγιστική ισορροπία Nash σε κάποιο σημείο (στρατηγικών των παικτών) όπου κανείς παίχτης δεν μπορεί να αυξήσει το κέρδος του περισσότερο από ε, επιλέγοντας μονομερώς να αλλάξει στρατηγική και να παίξει μια διαφορετική όταν οι άλλοι παίκτες διατηρούν τις στρατηγικές τους.

Πιο φορμαλιστικά:

Ορισμός της κατά προσέγγιση Ισορροπία Nash (ε - Nash Equilibrium)
Για κάθε ε>0, ένα ζεύγος στρατηγικών $(\tilde{x}, \tilde{y})$ είναι προσεγγιστική ισορροπία Nash για ένα διπαίγνιο, όταν ισχύει ότι:

1) Για κάθε πιθανή μικτή στρατηγική x του παίκτη γραμμής: $x^T R\tilde{y} \le \tilde{x}^T R\tilde{y} + \varepsilon$
   και
2) Για κάθε πιθανή μικτή στρατηγική y του παίκτη στήλης: $\tilde{x}^T Cy \le \tilde{x}^T C\tilde{y} + \varepsilon$

Που σημαίνει στην περίπτωση (1) ότι αν x η πιθανοτική κατανομή στρατηγικών στην οποία μονομερώς αλλάζει ο παίκτης γραμμής, τότε το κέρδος που θα έχει αν παίξει αυτήν, όποια κι αν είναι αυτήν, θα είναι πάντα μικρότερο ή ίσο με ε.
Ομοίως για την περίπτωση (2), για τον παίκτη στήλης.
Άρα δεν συμφέρει σε κανέναν από τους δύο παίκτες να αποκλίνει από την στρατηγική $(\tilde{x}, \tilde{y})$ αντίστοιχα, αν θέλει να αυξήσει το κέρδος του παραπάνω από ε.

Ορισμός ε -well – Nash Equilibria
Υπάρχει ακόμη μια παραλλαγή της προσεγγιστικής ισορροπίας Nash, την οποία δεν θα μελετήσουμε στην εργασία αυτήν, αλλά αξίζει να αναφέρουμε μιας και είναι αρκετά παρεμφερής με αυτήν που μελετούμε και έχει εξεταστεί και ερευνηθεί εξίσου με τις απλές προσεγγιστικές ισορροπίες Nash.

Για κάθε ε>0, ένα ζεύγος στρατηγικών $(\tilde{x}, \tilde{y})$, τέτοιο ώστε ο κάθε παίκτης παίζει μόνο προσεγγιστικές best-response αγνές στρατηγικές μη μηδενικής πιθανότητας.
είναι ε -well – Nash Equilibria για ένα διπαίγνιο, όταν ισχύει ότι:

1) Για κάθε πιθανή μικτή στρατηγική $x_i \succ 0$ του παίκτη γραμμής:
   $x_i^T R\tilde{y} \le x_j^T R\tilde{y} + \varepsilon$, $\forall j$
   και
2) Για κάθε πιθανή μικτή στρατηγική $y_i \succ 0$ του παίκτη στήλης:
   $\tilde{x}^T Cy_i \le \tilde{x}^T Cy_j + \varepsilon$, $\forall j$

Κάθε ε - well – Nash Equilibria είναι επίσης και ε – Nash Equilibria, αλλά όχι το αντίθετο.

2.8 PPAD και Εύρεση Ισορροπία Nash

Ένα από τα κυρίαρχα ζητήματα της Αλγοριθμικής Θεωρίας Παιγνίων είναι το πόσο υπολογιστικά δύσκολο είναι να βρεθεί μια ισορροπία Nash. Παρακάτω θα αναλυθεί η κλάση η οποία περιέχει το πρόβλημα της εύρεσης ισορροπίας Nash και ονομάζεται PPAD.

Η κλάση PPAD (Polynomial Parity Arguments on Directed graphs) αποτελεί μια κλάση πολυπλοκότητας που πρώτος ο Χ. Παπαδημητρίου εισήγαγε το 1994.

Η κλάση PPAD είναι υποσύνολο της κλάσης TFNP, η οποία είναι υποκλάση της FNP, η οποία περιλαμβάνει όλα τα προβλήματα εύρεσης (“NP-search problems") για τα οποία ξέρουμε ότι υπάρχει λύση.

Η κλάση FNP
Η κλάση FNP (Fully Non Polynomial) είναι η κλάση εκείνη στην οποία έχουμε να κάνουμε με ένα πρόβλημα απόφασης (το οποίο απαντά με ένα «ναι» ή ένα «όχι»), αλλά στην περίπτωση της απάντησης «ναι», απαιτείται η επιστροφή της λύσης του προβλήματος. Η κλάση FNP είναι τουλάχιστον όσο δύσκολη είναι η κλάση NP.

Η κλάση TFNP
Η κλάση TFNP (Total Function Non Polynomial) όπως είπαμε είναι μια υποκλάση της FNP. Τα προβλήματα που εμπεριέχει είναι αυτά στα οποία γνωρίζουμε ότι η απάντηση στο πρόβλημα είναι «ναι» και απλά πρέπει να επιστραφεί μία λύση.
Η σχέση που υπάρχει μεταξύ της κλάσης TFNP και της NP δεν έχει προσδιοριστεί ακόμα.
Πιο φορμαλιστικά η κλάση TFNP ορίζεται ως εξής:

> Μια δυϊκή σχέση P(x,y) που βρίσκεται στην TFNP αν και μόνο αν υπάρχει ντετερμινιστικό αυτόματο το οποίο σε πολυωνυμικό χρόνο μπορεί να αναγνωρίσει αν η P(x,y) σταματά (δηλαδή να δίνει απάντηση) δοθέντος τα x και y και επίσης για κάθε x, υπάρχει τέτοιο y που η P(x,y) να σταματά.

Όπως χαρακτηριστικά αναφέρει και ο Χρήστος Παπαδημητρίου η TFNP είναι μια εννοιολογική κλάση, με την έννοια ότι δεν είναι complete. Δεν εμπεριέχει προβλήματα στα οποία να μπορώ να κάνω αναγωγή από άλλα προβλήματα.
Το γεγονός αυτό δημιουργεί την ανάγκη εύρεσης υποκλάσεων της TFNP που να εμπεριέχουν τέτοιου είδους προβλήματα. Μια τέτοια κλάση είναι η PPAD (που είναι επίσης υποκλάση της PPA).

Η κλάση PPAD
Η PPAD είναι υποκλάση της TFNP, όπου η πιστοποίηση ότι υπάρχει τέτοιο y που η P(x,y) να σταματά στηρίζεται σε ένα parity argument ενός κατευθυνόμενου γράφου.

Ορισμός Parity Argument
Κάθε γράφος μέγιστου βαθμού 2 έχει άρτιο αριθμό από κόμβους βαθμού 1 ή διαφορετικά έχει άρτιο αριθμό φύλλων.

Όπως αναφέραμε ο John Nash απέδειξε ότι σε κάθε παίγνιο υπάρχει πάντα μια μικτή ισορροπία Nash [4] [5].
Στην πρώτη δημοσίευσή του απέδειξε το παραπάνω χρησιμοποιώντας το θεώρημα σταθερού σημείου του Kakutani, ενώ στην δεύτερη του δημοσίευση χρησιμοποίησε το θεώρημα σταθερού σημείου του Brouwer.
Αποδείχθηκε ότι η εύρεση μιας ισορροπίας Nash είναι ισοδύναμη με την εύρεση ενός σταθερού σημείου μιας συνάρτησης που μοντελοποιεί το εκάστοτε παίγνιο. Μια συνάρτηση Brouwer δύναται να μοντελοποιήσει ένα παίγνιο έτσι ώστε τα σταθερά σημεία της να είναι οι ισορροπίες Nash του παιγνίου.

Ήδη από το 1964 οι Lemke και Howson πρότειναν έναν αλγόριθμο εύρεσης μίας ισορροπίας Nash σε διπαίγνιο (bimatrix game).
Αποδείχθηκε ότι ο αλγόριθμος αυτός απαιτεί εκθετικό αριθμό βημάτων, ανεξάρτητα από το σημείο εκκίνησης, ακόμα και για win-loose παίγνια.

Το πρόβλημα της εύρεσης N.E. (Nash Equilibria) δεν μπορεί να αποδειχθεί ότι είναι NP-complete, μιας και ξέρουμε ήδη ότι υπάρχει τουλάχιστον μία, στην οποία το πρόβλημα απόφασης δίνει πάντα και τετριμμένα απάντηση ύπαρξης. Η απόδειξη ύπαρξης αυτή δεν είναι κατασκευαστική.
Βλέπουμε λοιπόν ότι το πρόβλημα εύρεσης ενός NE έχει “inefficient proof of existence", κάτι που το τοποθετεί στην κλάση PPAD.
Μέχρι πρόσφατα δεν ήταν γνωστό αν το πρόβλημα της εύρεσης N.E. είναι πλήρες για την κλάση PPAD. Τελικά αποδείχθηκε ότι ακόμα και για παίγνια με 4, 3 και 2 παίκτες το πρόβλημα είναι PPAD-complete.

Τέλος να τονίσουμε ότι, παρόλο που το πρόβλημα εύρεσης ενός NE δεν είναι NP-complete, το πρόβλημα απόφασης της ύπαρξης περισσοτέρων του ενός NE σε ένα παίγνιο, είναι.

Κεφάλαιο 3

Γραμμικός Προγραμματισμός

3.1 Ορισμός Γραμμικού Προγραμματισμού

Γραμμικός προγραμματισμός είναι η διαδικασία εύρεσης μιας βέλτιστης λύσης μιας γραμμικής συνάρτησης (αντικειμενικής συνάρτησης), η οποία ισχύει κάτω από ένα πεπερασμένο σύνολο γραμμικών ανισοτήτων (περιορισμοί).
Ο γραμμικός προγραμματισμός είναι κάτι το οποίο χρησιμοποιήθηκε από αρκετά παλιά ως μέθοδος επίλυσης προβλημάτων, μιας και πολλά προβλήματα ανάγονται σε γραμμικά προγράμματα. Όπως θα δούμε παρακάτω ένα τέτοιο πρόβλημα που ανάγεται σε γραμμικό πρόγραμμα είναι και αυτό που καλούμαστε να λύσουμε εμείς (της εύρεσης προσεγγιστικής ισορροπίας Nash).

Παράδειγμα 3.1

Το πρόβλημα της ΔΙΑΙΤΑΣ.
Έχοντας στην διάθεσή μας ένα πλήθος προϊόντων θα υποθέσουμε ότι αυτά μας παρέχουν Θερμίδες και Πρωτεΐνες. Επίσης ανάλογα την ποσότητα, τα προϊόντα έχουν ένα κόστος σε Ευρώ.
Υποθέτουμε ότι θέλουμε να σε καθημερινή βάση να προσλαμβάνουμε μια ελάχιστη ποσότητα θερμίδων και πρωτεϊνών, προσπαθώντας να δαπανήσουμε το ελάχιστο δυνατό ποσό (σε Ευρώ) κάθε μέρα.

Το πρόβλημα περιγράφεται στον παρακάτω πίνακα:

Είδος τροφής	**Θερμίδες**	**Πρωτεΐνες**	**Τιμή [Ευρώ]**
Κοτόπουλο	*205*	*32*	*2,4*
Γάλα	*160*	*8*	*0,9*
Χοιρινό	*260*	*14*	*1,9*
Απαιτήσεις ανθρώπου	**2000**	**55**	

Η τελευταία γραμμή απεικονίζει τις απαιτήσεις που έχει ένας άνθρωπος καθημερινά.

Περιορισμοί
Ορίζουμε ως x_i την δόση (την ποσότητα) της i-στης τροφής.
Οι περιορισμοί μπορούν να περιγραφθούν από τις παρακάτω ανισότητες:
$205 * x_1 + 160 * x_2 + 260 * x_3 \geq 2000$
$32 * x_1 + 8 * x_2 + 14 * x_3 \geq 55$
καθώς και ότι πρέπει οι ποσότητες $x_1, x_2, x_3 \geq 0$, αφού μιλάμε για υλικά αγαθά.

Αντικειμενική Συνάρτηση
Όπως είπαμε η αντικειμενική συνάρτηση ορίζει την ελαχιστοποίηση την συνολικής τιμής που θα πληρώσουμε σε Ευρώ.

Άρα θέλουμε:
minimize: $2.4*x_1+0.9*x_2+2.9*x_3$

Οι παραπάνω περιορισμοί και η αντικειμενική συνάρτηση ορίζουν ένα γραμμικό πρόγραμμα. Η λύση του οποίου θα δώσει τιμές στις μεταβλητές x_1, x_2, x_3 έτσι ώστε η τιμή της αντικειμενικής συνάρτησης να γίνει η ελάχιστη δυνατή που ικανοποιεί τους περιορισμούς.

-o-

3.2 Στοιχεία Πολυπλοκότητας

Μια πολύ σημαντική κλάση προβλημάτων ανάγονται σε Γραμμικό Προγραμματισμό. Αυτό είναι πολύ σημαντικό όσον αφορά την Θεωρητική Πληροφορική.
Το 1947 προτάθηκε ο αλγόριθμος SIMPLEX από τον George Dantzig, μέσω του οποίου λύνονται γραμμικά προγράμματα. Παρ` όλα αυτά στην χειρότερη περίπτωση έχει εκθετικό χρόνο εκτέλεσης. Στην πράξη βέβαια είναι πάρα πολύ αποδοτικός.

Ήδη από πολύ νωρίς αποδείχθηκε ότι ο γραμμικός προγραμματισμός ανήκει στην κλάση $NP \cap co-NP$. Παρ` όλα αυτά δεν υπήρχε κανείς γνωστός αλγόριθμος που να χρειάζονταν πολυωνυμικό χρόνο για να λύσει ένα γραμμικό πρόγραμμα.
Στην δεκαετία του 1970 ο Ελλειψοειδής αλγόριθμος, ήταν ο πρώτος που έτρεχε σε πολυωνυμικό χρόνο, ενώ την δεκαετία του 1980 ανακαλύφθηκε ο αλγόριθμο του Καρμακάρ, ο οποίος εισήγαγε την μελέτη των μεθόδων εσωτερικών σημείων για τον γραμμικό προγραμματισμό.

3.3 Γραμμικό Πρόγραμμα

Γραμμικό Πρόγραμμα είναι το πρόβλημα της μεγιστοποίησης μια συνάρτησης ωφέλειας ή αντίστοιχα της ελαχιστοποίησης μιας συνάρτησης κόστους (αντικειμενικής συνάρτησης), υπό κάποιες συνθήκες που ονομάζονται περιορισμοί.
Η συνάρτηση αυτή εξαρτάται από ένα σύνολο μεταβλητών απόφασης $x_1, x_2, ..., x_n$ με την προϋπόθεση ότι τηρούνται κάποιοι περιορισμοί ως προς τις τιμές των μεταβλητών αυτών. Αυτοί οι περιορισμοί εκφράζονται μέσα από ένα σύνολο γραμμικών ισοτήτων και ανισοτήτων.

$$\boxed{\min\{c^T x : Ax = b; A`x \geq b`; x_j \geq 0, \forall j \in I \subseteq [n]\}} \quad \text{-} \quad \textbf{LP Problem}$$

όπου:
$x \in \mathbb{R}^{nx1}, b \in \mathbb{R}^{mx1}, b` \in \mathbb{R}^{m`x1}, c \in \mathbb{R}^{nx1}$

$$A = \begin{pmatrix} a_{11} & \cdots & a_{n1} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{nm} \end{pmatrix} \in \mathbb{R}^{mxn}$$

$$A` = \begin{pmatrix} a`_{11} & \cdots & a`_{n1} \\ \vdots & \ddots & \vdots \\ a`_{m1} & \cdots & a`_{nm} \end{pmatrix} \in \mathbb{R}^{m`xn}$$

Εφικτή Λύση
Αυτό που αναζητούμε ουσιαστικά στο παραπάνω LP είναι το διάνυσμα x (δηλαδή τις μεταβλητές απόφασης) που να ικανοποιεί την σχέση που είδαμε.
Ένα τέτοιο πραγματικό διάνυσμα x που ικανοποιεί του περιορισμούς και την μη-αρνητικότητα του LP θα το ονομάζουμε εφικτή λύση (feasible solution).

Μη Επιλύσιμο Πρόγραμμα
Μη επιλύσιμο (infeasible) θα ονομάζεται εκείνο το LP που δεν έχει καμιά εφικτή λύση.

Βέλτιστη Λύση
Ένα διάνυσμα x* που εκτός από τους του περιορισμούς και την μη-αρνητικότητα του LP, δίνει την μικρότερη τιμή στην αντικειμενική συνάρτηση (εφόσον μιλάμε για minimization) ή την μεγαλύτερη τιμή στην αντικειμενική συνάρτηση (εφόσον μιλάμε για maximization), δηλαδή βελτιστοποιεί την αντικειμενική συνάρτηση, θα την ονομάζουμε βέλτιστη λύση.

Υπάρχουν πολλοί τρόποι αναπαράστασης γραμμικών προγραμμάτων από τις οποίες θα δούμε μόνο αυτήν που θα χρησιμοποιήσουμε εμείς (κανονική μορφή).

3.4 Μορφές Γραμμικών Προγραμμάτων

Κάθε LP μπορεί να αναπαρασταθεί με διαφορετικούς τρόπους. Βέβαια εμείς θα δουλέψουμε με την κανονική μορφή που θα περιγράψουμε παρακάτω, αλλά πλέον είναι τετριμμένη η μετατροπή κάθε μορφής σε οποιαδήποτε άλλη, μέσω κάποιων κανόνων.

Γενική Μορφή (General Form):
Είναι το πρόβλημα της ελαχιστοποίησης μια γραμμικής συνάρτησης (αντικειμενική συνάρτηση), βάσει ενός πεπερασμένου αριθμού γραμμικών περιορισμών ισότητας και ανισότητας.

$$\min\{c^T x : Ax = b; A`x \geq b`; x_j \geq 0, \forall j \in I\} \quad \textbf{-} \quad \textbf{G-LP}$$

όπου υπάρχει συνδυασμός περιορισμών ισότητας και ανισότητας και ο περιορισμός στο πρόσημο των μεταβλητών δεν είναι περιοριστικός για όλες τις μεταβλητές. Μπορούν να υπάρξουν μεταβλητές που να μην έχουν κανέναν περιορισμό όσο αφορά το πρόσημό τους.

Τυπική Μορφή (Standard Form):

$$\min\{c^T x : Ax = b; x \geq 0\} \quad \textbf{-} \quad \textbf{S-LP}$$

Υπάρχει μόνο ο περιορισμός ισότητας, ενώ οι μεταβλητές έχουν όλες μη-αρνητικό πρόσημο.

Κανονική Μορφή (Canonical Form):

$$\min\{c^T x : Ax \geq b; x \geq 0\} \quad \textbf{-} \quad \textbf{C-LP}$$

Επιτρέπονται μόνο οι περιορισμοί ανισότητας και μη αρνητικές τιμές στις μεταβλητές του προβλήματος.

Και οι τρεις αυτές μορφές είναι ισοδύναμες και θα δούμε παρακάτω με ποιους κανόνες μπορούμε να δημιουργήσουμε την μία από την άλλη.

3.5 Μετατροπές

Οι μετατροπές των διάφορων μορφών σε άλλες, είναι πλέον τετριμμένες. Μπορούμε για παράδειγμα ένα πρόβλημα μεγιστοποίησης να το μετατρέψουμε σε πρόβλημα ελαχιστοποίησης, αλλάζοντας απλά το πρόσημο του διανύσματος των συντελεστών της αντικειμενικής συνάρτησης. Κάτι τέτοιο θα το κάνουμε στην εργασία αυτήν όταν εφαρμόζουμε τον αλγόριθμο. (μετατροπή του minimax προβλήματος σε min)
Ισοδύναμες θα είναι εκείνες οι μορφές LP στις οποίες υπάρχει το ίδιο σύνολο βέλτιστων λύσεων ή είναι και οι δύο μορφές μη-επιλύσιμες.

Μετατροπή μορφής ελαχιστοποίησης σε μορφή μεγιστοποίηση
Απλώς αλλάζουμε το πρόσημο των συντελεστών της συνάρτησης.

$$\min\{c^T x : Ax = b; x \geq 0\} \Leftrightarrow \max\{-c^T x : Ax = b; x \geq 0\}$$

Μετατροπή περιορισμών ισότητας σε περιορισμούς ανισότητας
Για κάθε i-στο περιορισμό οι περιορισμοί μπορούν να μετατραπούν εύκολα από την μία μορφή στην άλλη βάσει του εξής κανόνα:

$$\forall i \in [m], A_i x = b_i \Leftrightarrow \{A_i x \leq b_i AND A_i x \geq b_i\} \Leftrightarrow \{A_i x \leq b_i AND -A_i x \leq -b_i\}$$

Μετατροπή περιορισμών ανισότητας σε περιορισμούς ισότητας
Η μετατροπή αυτή γίνεται μέσω αδιάφορων μεταβλητών s:

$$\forall i \in [m], A_i x = b_i \Leftrightarrow \{A_i x + s_i \leq b_i, s_i \geq 0\}$$

Μετατροπή μη θετικότητας σε μη αρνητικότητα
Απλώς αντικαθιστούμε την μεταβλητή $x_j \leq 0$ με την $xy = -x_j \geq 0$ στο LP μας.

Μετατροπή μιας κανονικής μορφής σε τυπική μορφή
Κι αυτό γίνεται μέσω αδιάφορων μεταβλητών s:
$$\min\{c^T x : Ax = b; x \geq 0\} \Leftrightarrow \min\{c^T x : Ax - Is = b; x, s \geq 0\}$$

Με άλλα λόγια και οι τρεις μορφές (Γενική , Τυπική, Κανονική) είναι ισοδύναμες, μέσω των μετατροπών που παρουσιάσαμε και χωρίς να γίνει κάποια μεγάλη αλλαγή στον αριθμό των μεταβλητών και των περιορισμών.

3.6 Προϋποθέσεις εφαρμογής του γραμμικού προγραμματισμού

Οι απαιτούμενες προϋποθέσεις για την εφαρμογή του γραμμικού προγραμματισμού σ' ένα πρόβλημα βελτιστοποιήσεως, μέσω των οποίων περιορίζεται το φάσμα της εφαρμογής του γραμμικού προγραμματισμού σε προβλήματα είναι:

1. **Γραμμικότητα**
   Όλες οι συναρτήσεις του προβλήματος, αντικειμενική συνάρτηση και περιορισμοί, πρέπει να είναι γραμμικές ως προς τις άγνωστες μεταβλητές. Δηλαδή να ισχύουν οι ιδιότητες της αναλογικότητας και της προσθετικότητας όσο αφορά τις μεταβλητές του προβλήματος.

2. **Διαιρετότητα**
   Το μοντέλο του γραμμικού προγραμματισμού υποθέτει ότι κάθε μεταβλητή είναι συνεχής και επομένως άπειρα διαιρετή. Αυτό συνεπάγεται ότι όλες οι μεταβλητές επιτρέπεται να πάρουν κλασματικές ή ακέραιες τιμές.

3. **Βεβαιότητα**
   Το μοντέλο του γραμμικού προγραμματισμού προϋποθέτει ότι όλοι οι παράμετροι του προβλήματος είναι γνωστές με απόλυτη βεβαιότητα.

Όπως είναι ορατό, οι προϋποθέσεις αυτές περιορίζουν το φάσμα εφαρμογής του Γραμμικού Προγραμματισμού σε διάφορα προβλήματα. Παρ` όλα αυτά υπάρχουν τρόποι αναγωγής των προβλημάτων που δεν ικανοποιούν τις προϋποθέσεις αυτές σε Γραμμικά Προγράμματα τα οποία δίνουν λύσης που προσεγγίζουν την επιθυμητή λύση.

3.7 Convex και Concave Συνάρτηση και Σύνολο

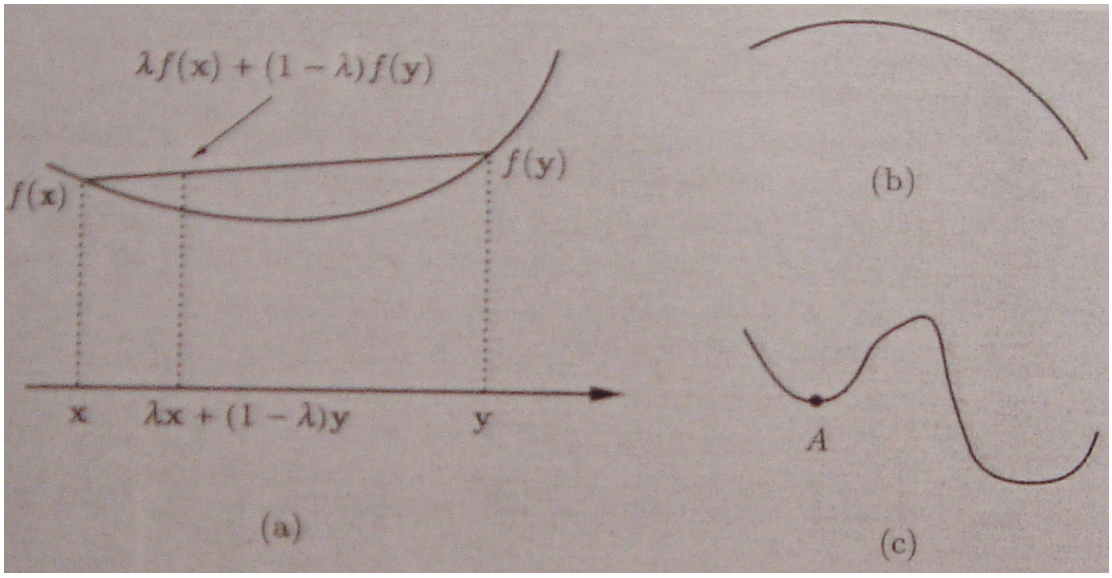
Συναρτήσεις

Μια συνάρτηση $f:\mathbb{R}^n \to \mathbb{R}$ ονομάζεται **convex** αν για κάθε $x, y \in \mathbb{R}^n$ και κάθε $\lambda \in [0,1]$ έχουμε:

$$f(\lambda x + (1-\lambda)y) \le \lambda f(x) + (1-\lambda) f(y)$$

Αν ισχύει η αντίθετη ανισότητα τότε ονομάζεται **concave**.

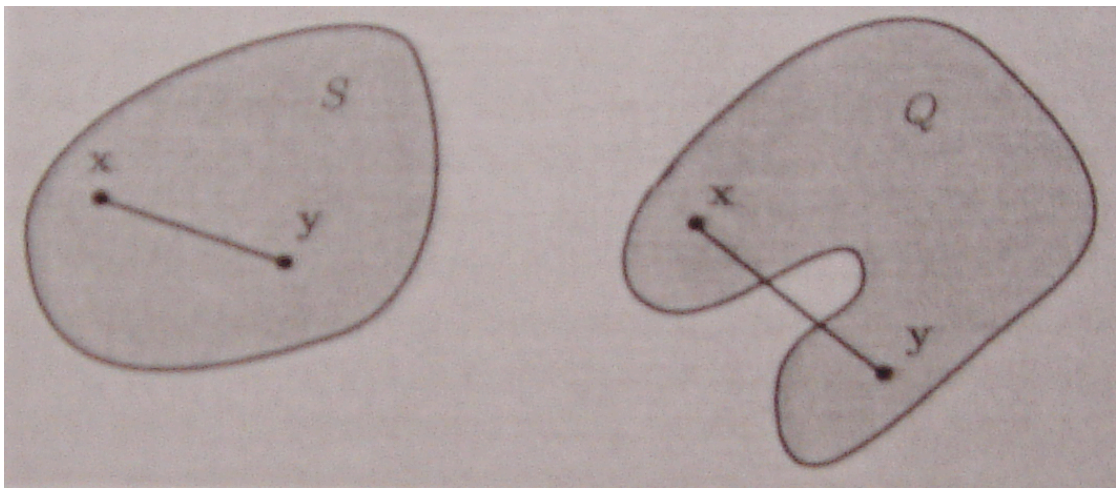
Αν οι συναρτήσεις $f_1, f_2, ..., f_m : \mathbb{R}^n \to \mathbb{R}$ είναι convex τότε η $f(x) = \max\{f_1, f_2, ..., f_m\}$ είναι επίσης convex.



Εικόνα 1: (a) Μία συνάρτηση convex, (b) Μία συνάρτηση concave, (c) Μια συνάρτηση που δεν είναι ούτε convex, ούτε concave.
Πηγή: [6]

Σύνολα

Ένα set $S \subset \mathbb{R}^n$ είναι **convex** αν για κάθε $x, y \in S$ και για κάθε $\lambda \in [0,1]$, έχουμε:
$\lambda x + (1-\lambda) y \in S$



Εικόνα 2: (a) Ένα convex set, (b) Ένα set που δεν είναι convex
Πηγή: [6]

3.8 Πολύεδρο – Vertex

Πολύεδρο

Πολύεδρο είναι ένα σύνολο που μπορεί να περιγραφθεί από την εξής σχέση:
$P = \{x \in \mathbb{R}^n : Cx \geq d\}$,
όπου το C είναι ένα πίνακας m x n και το b είναι διάνυσμα μέσα στο $\mathbb{R}^m$.

Εφικτή Λύση

Έστω M, N πεπερασμένα σύνολα και $I \subseteq M$ και $J \subseteq N$, τότε για κάποια δεδομένα διανύσματα $b \in \mathbb{R}^M$, $c \in \mathbb{R}^N$ και κάποιον πίνακα $A \in \mathbb{R}^{MxN}$, θεωρούμε το πολύεδρο:

$$P = \begin{cases} x \in \mathbb{R}^N \mid \sum_{j \in N} a_{ij} x_j = b_i, i \in M - I \\ \sum_{j \in N} a_{ij} x_j \le b_i, i \in I \\ x_j \ge 0, j \in J \end{cases}$$

Κάθε διάνυσμα x που ανήκει στο πολύεδρο P ονομάζεται **εφικτή λύση** του πολύεδρου αυτού.
Ένα γραμμικό πρόγραμμα αναζητά το διάνυσμα x που είναι ταυτόχρονα εφικτή λύση και βέλτιστη λύση της αντικειμενικής συνάρτησης.

Κορυφή (vertex)
Το σημείο $x \in P$ ονομάζεται κορυφή (vertex) του P αν και μόνο αν δεν μπορεί να γραφτεί ως αυστηρά κυρτός συνδυασμός δύο άλλων διανυσμάτων του P.

Πιο φορμαλιστικά:

$$\Delta \varepsilon \nu \ \upsilon \pi \acute{\alpha} \rho \chi \varepsilon \iota \ \ y, z \in P \setminus \{x\}, \lambda \in (0,1) : x = \lambda y + (1 - \lambda) z$$

Δηλαδή αν x είναι κορυφή, τότε δεν μπορούμε να κινηθούμε κατά μήκος ενός ευθυγράμμου τμήματος που ενώνει δύο σημεία του P και περνά από το x, εφόσον το x δεν είναι κάποιο από τα άκρα του.

3.9 Βασική Λύση

Στην τυπική μορφή του προβλήματος αν το σύστημα έχει r μεταβλητές και m γραμμικές εξισώσεις και ισχύει ότι m<r , τότε βασική λύση είναι μία λύση που προκύπτει αν θέσουμε r-m μεταβλητές ίσες με το μηδέν και λύσουμε ως προς τις υπόλοιπες m μεταβλητές υπό τον όρο το γραμμικό σύστημα που προκύπτει να περιλαμβάνει m γραμμικώς ανεξάρτητες στήλες του πίνακα A.
Δηλαδή ο πίνακας A να περιέχει έναν υποπίνακα διάστασης m x m για τον οποίο να ισχύει ότι η ορίζουσα του είναι διάφορη του μηδενός.

Η τετραγωνική μήτρα, που προκύπτει από τον A και έχει m γραμμικά ανεξάρτητες στήλες καλείται **βάση** του συστήματος, ενώ οι m μεταβλητές που αντιστοιχούν στις στήλες της βάσεως καλούνται **βασικές μεταβλητές**. Οι υπόλοιπες r-m μεταβλητές που δεν περιλαμβάνονται στη βάση καλούνται **μη βασικές μεταβλητές**. Αν συμβολίσουμε τον υποπίνακα των m γραμμικά ανεξάρτητων στηλών του A με B, (det(B) ≠ 0), η βασική λύση μπορεί να υπολογιστεί από την σχέση:
$x = B^{-1} b$, όπου το x είναι το διάνυσμα των βασικών μεταβλητών.

Ο πίνακας B ονομάζεται και **βασικός πίνακας**.

Βασική εφικτή λύση είναι μία βασική λύση όπου όλες οι βασικές μεταβλητές είναι μη αρνητικές.

Μη εκφυλισμένη βασική εφικτή λύση είναι μία βασική εφικτή λύση με ακριβώς m θετικά x_i και έχει λιγότερα από m θετικά x_i.

3.10 Αλγόριθμοι επίλυσης SIMPLEX (σημείο εκκίνησης και απόδοση)

Ο αλγόριθμος SIMPLEX επιλύει ένα γραμμικό πρόγραμμα εστιάζοντας στις βασικές εφικτές λύσεις του. Στην χειρότερη περίπτωση του δεν είναι πολυωνυμικός, αλλά πρακτικά είναι αρκετά γρήγορος.

Λειτουργία Αλγορίθμου

Γνωρίζοντας πώς να ελέγξουμε αν ένα γραμμικό πρόγραμμα είναι πεπερασμένο ή όχι, τότε αν η συνάρτηση κόστους (αντικειμενική συνάρτηση) ενός γραμμικού προγράμματος έχει κάτω φράγμα στο χώρο λύσεων, αρκεί να ελέγξουμε όλες τις εκθετικές σε αριθμό κορυφές του πολυέδρου των λύσεων, για να βρούμε βέλτιστη λύση.
Ο SIMPLEX εξετάζει τις κορυφές του πολυέδρου. Όχι όλες. Ξεκινάει από μια κορυφή και τη συγκρίνει με τις γειτονικές της κορυφές. Αν μεταξύ αυτών υπάρχει κάποια που να δίνει καλύτερη λύση τότε μετακινείται σε αυτήν. Αυτή η διαδικασία συνεχίζεται μέχρι να βρεθεί κάποια κορυφή που να είναι η καλύτερη δυνατή.

Πρακτικά Αποτελέσματα

Στην πράξη ο αλγόριθμος SIMPLEX δεν κάνει εκθετικό αριθμό βημάτων.
Έχει δειχθεί στην πράξη ότι ο αλγόριθμος κάνει 4m με 6m εναλλαγές και σπάνια πάνω από 10m βήματα, όπου m είναι ο αριθμός κορυφών. Επίσης καθώς μεγαλώνει ο αριθμός m των μεταβλητών του προβλήματος, ο αριθμός των εναλλαγών αυξάνεται με πολύ αργό ρυθμό.

3.11 Δυϊκό Γραμμικό Πρόγραμμα (Dual LP)

Ο γραμμικός προγραμματισμός και ιδιαίτερα η δυϊκότητα (δηλαδή η σύνδεση του primal προβλήματος με το δυϊκό), αποτελούν ισχυρά αποδεικτικά εργαλεία, τα οποία έχουν σωρεία εφαρμογών στον γραμμικό προγραμματισμό. Χρησιμοποιήθηκαν περισσότερα στους προσεγγιστικούς αλγόριθμους. Μέσω της δυϊκότητας ήρθε στο φως μια νέα μέθοδος επίλυσης γραμμικών προγραμμάτων, την οποία χρησιμοποιεί το βοηθητικό πρόγραμμα CPLEX που χρησιμοποιούμε στην παρούσα εργασία για την λύση των LPs που έχουμε και ονομάζεται dual simplex method.

Ένα (primal) γραμμικό πρόγραμμα μπορεί να συνδεθεί με ένα άλλο (dual) γραμμικό πρόγραμμα, ακολουθώντας τους εξής μηχανισμούς:

Primal		**Dual**	
Minimize	$c`x$	Maximize	$p`b$
Subject to	$a`_i x \geq b_i$	Subject to	$p_i \geq 0$
	$a`_i x \leq b_i$		$p_i \leq 0$
	$a`_i x = b_i$		$p_i free$
	$x_j \geq 0$		$p`A_j \leq c_j$
	$x_j \leq 0$		$p`A_j \geq c_j$

	$x_j = 0$		$p`A_j = c_j$

Έτσι δημιουργείται από ένα dual γραμμικό πρόβλημα, ένα ισοδύναμο dual.
Κάθε dual μεταβλητή σχετίζεται με έναν συγκεκριμένο primal περιορισμό, που αντιπροσωπεύει την τιμωρία (penalty) για την παραβίαση του περιορισμού αυτού.
Αντικαθιστώντας τους primal περιορισμούς με τις dual μεταβλητές, αυξάνουμε το πλήθος τον δυνατών επιλογών και αυτό μας βοηθάει στο να δημιουργούμε primal λύσεις των οποίων το κόστος είναι μικρότερο από της καλύτερης λύσης που είχαμε βρει στο primal πρόβλημα.

Έτσι κάθε γραμμικό πρόγραμμα LP μπορεί να μετατραπεί σε ένα άλλο γραμμικό πρόγραμμα Dual LP.

Το αντίστοιχο πολύεδρο που δημιουργείται είναι το εξής:

$$D = \begin{cases} y \in \mathbb{R}^M \mid \sum_{i \in M} y_i a_{ij} = c_j, i \in N - J \\ \sum_{i \in M} y_i a_{ij} \le c_j, j \in J \\ y_i \ge 0, i \in I \end{cases}$$

Για ένα ζεύγος γραμμικών προγραμμάτων Primal LP και το αντίστοιχο Dual LP, ισχύει το Θεώρημα:

Αν πάρουμε ένα δυϊκό (dual) πρόγραμμα LP και βρούμε το ισοδύναμο ελάχιστο πρόβλημα και στη συνέχεια δημιουργήσουμε το δυϊκό αυτού του προγράμματος, είναι το ίδιο το αρχικό γραμμικό πρόγραμμα.

Θεώρημα Δυϊκότητας

1. (**Ασθενής Δυϊκότητα**) Αν x είναι η εφικτή λύση του primal προβλήματος ($x \in P$) και p η εφικτή λύση του dual προβλήματος ($p \in D$), τότε:
   $p`b \le c`x$
2. (**Ισχυρή Δυϊκότητα**) Αν ένα πρόβλημα γραμμικού προγράμματος έχει βέλτιστη λύση, τότε και το dual του έχει επίσης και οι τιμές των αντικειμενικών συναρτήσεων των δύο αυτών γραμμικών προγραμμάτων ταυτίζονται για τις βέλτιστες αυτές λύσεις.
   Δηλαδή αν $x \in P$ και $p \in D$ τότε $p`b = c`x$.

Για ένα ζεύγος δυϊκών μεταξύ τους προγραμμάτων ισχύει το παρακάτω θεώρημα.

Θεώρημα Συμπληρωματική Χαλαρότητα
Αν x μια εφικτή λύση του primal προβλήματος και p μια εφικτή λύση του dual προβλήματος, τότε τα x, p είναι βέλτιστες λύσεις των δύο αντίστοιχων προβλημάτων αν και μόνο αν:

$$p_i(a`_i x - b_i) = 0, \forall i$$
$$(c_j - p`A_j)x_j = 0, \forall j$$

Κεφάλαιο 4

Προηγούμενα αποτελέσματα για εύρεση Προσεγγιστικής Ισορροπίας Nash

Όπως είδαμε το πρόβλημα της εύρεσης ισορροπίας Nash σε πολυωνυμικό χρόνο ακόμα και για παίγνια δύο παικτών είναι ένα ανοιχτό πρόβλημα που δεν έχει επιλυθεί ακόμα.

4.1 Nash Equilibria

Δεν είναι ακόμα γνωστό αν μπορεί να υπολογιστεί αποδοτικά μια ισορροπία Nash. Για παίγνια με δύο παίκτες υπάρχουν ήδη γνωστοί αλγόριθμοι που παρουσιάζονται στα [15,16,17,18,19,20,21], αλλά είτε έχουν εκθετικό χρόνο (στο μέγεθος των δυνατών αγνών στρατηγικών) εκτέλεσης στην χειρότερη περίπτωση, είτε είναι άγνωστο αν τρέχουν σε πολυωνυμικό χρόνο.
Για παίγνια τριών παικτών το πρόβλημα είναι ακόμα πιο δύσκολο.
Ενώ το πρόβλημα εύρεσης ισορροπίας Nash για παίγνια δύο παικτών μπορεί να αναχθεί σε πρόβλημα LCP (Linear Complementarity Problem), το πρόβλημα με τρεις παίκτες είναι Non-LCP.

Πολλοί ερευνητές, όπως είπαμε, επικεντρώθηκαν στον υπολογισμό προσεγγιστικών ισορροπιών Nash, όπως αυτό που μελετάμε στην εργασία αυτήν.
Παρακάτω θα παρουσιάσουμε μερικά καλά αποτελέσματα για προσεγγιστικές ισορροπίες Nash (ε-approximate Nash Equilibria, με $\varepsilon \succ 0$), όπως αυτά εξελίχθηκαν βάσει ερευνών διαφόρων ομάδων που ασχολούνται με την αλγοριθμική θεωρία παιγνίων.

Ορισμός 4.1: ε-approximate Nash Equilibria
Όπως είπαμε και παραπάνω, ένα σετ μικτών στρατηγικών ονομάζεται ε-approximate Nash Equilibria για $\varepsilon \succ 0$, όταν για κάθε παίκτη όλες οι στρατηγικές έχουν αναμενόμενο κέρδος το πολύ ε περισσότερο από το αναμενόμενο κέρδος της δεδομένης στρατηγικής.

4.2 ε-approximate Nash Equilibria

Δεδομένου αυτού είναι προφανές ότι σε κανονικοποιημένους πίνακες κέρδους (δηλαδή οι τιμές των κερδών έχουν κανονικοποιηθεί μεταξύ 0 και 1), κάθε μικτή στρατηγική είναι 1-approximate Nash Equilibrium.

Επίσης μπορούμε να βρούμε μια $\frac{3}{4}$ - approximate Nash Equilibrium, αν εξετάσουμε όλα τα supports μεγέθους δύο.

Στο [10] δίνεται ένας τρόπος υπολογισμού $\frac{2+\varepsilon+\lambda}{4}$ - approximate Nash Equilibrium, για κάθε ε, όπου λ είναι το ελάχιστο αναμενόμενο κέρδος του κάθε παίκτη, μεταξύ όλων των ισορροπιών Nash που υπάρχουν. Ο προτεινόμενος αυτός αλγόριθμος τρέχει

σε πολυωνυμικό χρόνο στο $\frac{1}{\varepsilon}$ και στο μέγεθος του προβλήματος (δηλαδή στον αριθμό των στρατηγικών που είναι διαθέσιμα για κάθε παίκτη).

Στο [11] αποδείχθηκε ότι για κάθε $\varepsilon \succ 0$ μπορεί να βρεθεί ε-approximate Nash Equilibria σε χρόνο $O(n^{\frac{\log n}{\varepsilon^2}})$, εξετάζοντας supports μεγέθους $\frac{\log n}{\varepsilon^2}$.
Ουσιαστικά στο [11] αποδείχθηκε ότι για κάθε διπαίγνιο και για κάθε σταθερό $\varepsilon \succ 0$, υπάρχει ε-approximate Nash Equilibrium με λογαριθμικό (στον αριθμό n των δυνατών αγνών στρατηγικών) αριθμό support. Κάτι τέτοιο οδηγεί σε ένα αλγόριθμο υπολογισμού ε-approximate Nash Equilibrium σε ψευδό-πολυωνυμικό χρόνο ($n^{O(\ln n)}$).
Επίσης αποδείχθηκε ότι οι στρατηγικές κάθε παίκτη σε τέτοιες ισορροπίες είναι ομοιόμορφες πάνω σε μικρά supports.

Επίσης αποδείχθηκε στο [12] ότι ακόμα και σε zero-sum παίγνια, δεν υπάρχει αλγόριθμος που ελέγχοντας supports μεγέθους μικρότερου του logn, να βρίσκει approximate Nash Equilibrium καλύτερη από $\frac{1}{4}$.

Στο [14] αποδείχθηκε ότι το πρόβλημα του υπολογισμού μιας $\frac{1}{n^{\Theta(1)}}$ - approximate Nash Equilibrium ανήκει στην κλάση PPAD-complete και ότι για διπαίγνια είναι αδύνατο να έχουμε έναν αλγόριθμο που να τρέχει σε πλήρη πολυωνυμικό χρόνο (εκτός αν $PPAD \subseteq P$). Ωστόσο γίνεται η εικασία ότι είναι απίθανο η εύρεση μιας ε - approximate Nash Equilibrium να ανήκει στην κλάση PPAD-complete όταν έχουμε απολύτως σταθερό ε.

Πρόσφατα βρέθηκαν αλγόριθμοι υπολογισμού ε - approximate Nash Equilibrium για σταθερά $\varepsilon = \frac{3}{4}$ και $\varepsilon = \frac{1}{2}$ στα [13] και [25] αντίστοιχα, για γενικευμένα παίγνια δύο παικτών, με θετικά κανονικοποιημένους πίνακες κέρδους. Οι αλγόριθμοι αυτοί εξετάζουν μικρά supports για έναν ή δύο παίκτες αντίστοιχα.
Η απόδειξη για την εύρεση $\frac{1}{2}$ - approximate Nash Equilibrium σε παίγνιο με δύο παίκτες είναι σχετικά απλή. Για κάθε i στρατηγική του παίκτη-1, θεωρούμε j το best response του παίκτη-2 και k το best response του παίκτη-1 στο j. Τότε ο παίκτης-1 παίζει εξίσου ίσα τις στρατηγικές i και k , όταν ο παίκτης-2 παίζει την στρατηγική j.

Επίσης στο [24] υπάρχει ένα πολύ καλό αποτέλεσμα που σπάει το φράγμα του $\frac{1}{2}$ και εξασφαλίζει, σε πολυωνυμικό χρόνο ($O(n^{\frac{1}{\varepsilon^2}})$), 0.38 - approximate Nash Equilibrium για κανονικοποιημένα διπαίγνια, θεωρώντας ότι υπάρχουν supports αυθαίρετα μεγάλου μεγέθους.

Επίσης πρόσφατα στο [26] έγινε βελτίωση της παραπάνω ισορροπίας από 0.38 σε 0.36 - approximate Nash Equilibrium χρησιμοποιώντας τεχνικές γραμμικού

προγραμματισμού σε παίγνια δύο παικτών. Η τεχνική στηρίζεται σε ελαχιστοποίηση του σφάλματος κατά τον υπολογισμό της προσεγγιστικής ισορροπίας Nash, όπως αυτή προκύπτει για zero-sum παίγνια.

Στο [27] αποδείχθηκε ότι η εύρεση προσεγγιστικής ισορροπίας για γενικά προβλήματα για κάποιο $\varepsilon \succ 0$ που να τρέχει σε πλήρες πολυωνυμικό χρόνο, έχει την ίδια πολυπλοκότητα με το πρόβλημα εύρεσης ακριβής (exact) ισορροπία Nash.

Τέλος στην εργασία [1] την οποία μελετούμε πρακτικά σε αυτό το paper,παρουσιάζεται ο καλύτερος μέχρι στιγμής αλγόριθμος εύρεσης προσεγγιστικής ισορροπίας Nash. Περιγράφεται ο αλγόριθμος εύρεσης 0.3393 - approximate Nash Equilibrium μέσω γραμμικού προγραμματισμού, που όπως θα δούμε και στην συνέχεια στην πράξη είναι πάρα πολύ καλύτερος (της τάξης του 10^{-3} - approximate Nash Equilibrium)

4.3 Well-supported ε-approximate Nash Equilibria

Στο [13] παρουσιάζεται ένας αλγόριθμος εύρεσης ενός well-supported $\frac{2}{3}$-approximate Nash Equilibrium με μέγεθος support το πολύ 3, για win-loose παίγνια. Αυτό οδήγησε στην εύρεση ενός $\frac{5}{6}$- approximate Nash Equilibrium για κάθε είδος παίγνιο.

Επίσης στο [28] αποδείχθηκε ότι είναι δυνατός ο υπολογισμός well-supported 0.658 - approximate Nash Equilibrium για κανονικοποιημένα παίγνια δύο παικτών σε πολυωνυμικό χρόνο.

Κεφάλαιο 5

Επεξήγηση Κώδικα Υλοποίησης

Στο κεφάλαιο αυτό θα γίνει η παρουσίαση της υλοποίησης του αλγορίθμου που περιγράφεται στην εργασία "An Optimization Approach for Approximate Nash Equilibria" των Χ. Τσακνάκη και Π. Σπυράκη [1] και θα αναλυθεί ενδελεχώς κάθε βήμα αυτού, όπως εφαρμόστηκε στην υλοποίηση του στην παρούσα εργασία.

5.1 Γενικές Παρατηρήσεις και Συμβολισμοί

Παρακάτω παρουσιάζονται κάποιες γενικές παρατηρήσεις και κάποιοι συμβολισμοί μεγεθών που θα χρησιμοποιηθούν στην συνέχεια.

- Τα παίγνια που θα μελετήσουμε ονομάζονται διπαίγνια, διότι έχουμε πάντα δύο παίκτες. Τον παίκτη γραμμής (row player) και τον παίκτη στήλης (column player).

- Τα παίγνια θα αναπαρασταθούν σε Κανονική - Στρατηγική μορφή (με την μορφή πινάκων κέρδους R και C) και όχι σε Εκτατική μορφή.

  Για να οριστεί ένα παίγνιο σε Κανονική – Στρατηγική μορφή πρέπει απλά να καθορίσουμε το σύνολο των παικτών στο παίγνιο, το σύνολο των επιλογών που έχει κάθε παίχτης (δηλαδή το σύνολο των στρατηγικών του) και τον τρόπο με τον οποίο εξαρτώνται τα κέρδη των παικτών από τις επιλογές τους.

- Τα διανύσματα που αναφέρονται στο πρόγραμμα είναι όλα διανύσματα στήλης.

- Σε ολόκληρο το πρόγραμμα θα δουλέψουμε με double αριθμούς. Δηλαδή, με άλλα λόγια, όλοι οι υπολογισμοί θα γίνονται με double ακρίβεια.

- Θεωρούμε ότι οι m και n είναι ακέραιοι αριθμοί και συμβολίζουν τον αριθμό των αγνών στρατηγικών που μπορούν να παίξουν ο παίκτης γραμμής και ο παίκτης στήλης αντίστοιχα.

- Στα προβλήματα που θα μελετήσουμε θα χρησιμοποιήσουμε τους πίνακες R και C για να περιγράψουμε τους πίνακες κέρδους του παίκτη γραμμής και στήλης αντίστοιχα. Αυτοί θα είναι μεγέθους m x n και θα περιέχουν double αριθμούς.

5.2 Περιγραφή του input, output και των ενδιάμεσων αποτελεσμάτων του προγράμματος

5.2.1 Τα στοιχεία που δίνουμε ως Είσοδο στο πρόγραμμα

Τα στοιχεία που δίνουμε ως είσοδο στο πρόγραμμα είναι αυτά που καθορίζουν το είδος και την μορφή του παιγνίου του οποίου την κατά προσέγγιση ισορροπία Nash θα αναζητήσουμε μέσω του προγράμματος.

Παρακάτω παρουσιάζονται τα τρία στοιχεία που θεωρούνται στοιχεία Εισόδου για το πρόγραμμα και δίνονται από τον χρήστη ανάλογα με το παίγνιο που θέλει να δημιουργήσει.

- Οι διαστάσεις του προβλήματος m και n
  (επίσης αναφέρονται ως row_size και col_size αντίστοιχα):

  Το **m** είναι ο ακέραιος αριθμός που εκφράζει τον αριθμό των στρατηγικών που μπορεί να παίξει ο παίκτης γραμμής (θα λέμε επίσης ότι m είναι αριθμός των γραμμών)
  και ο **n** ο ακέραιος που εκφράζει τον αριθμό των στρατηγικών που μπορεί να παίξει ο παίκτης στήλης (θα λέμε επίσης ότι m είναι αριθμός των στηλών).

  Ουσιαστικά θα λέμε ότι **m x n** είναι μέγεθος ενός παιγνίου.

- Πίνακες R, C:
  Οι πίνακες **R** και **C** είναι οι πίνακες κέρδους (payoff matrices) του παίκτη γραμμής και στήλης αντίστοιχα.
  Τα στοιχεία που έχουν οι πίνακες είναι δεκαδικοί αριθμοί, double ακρίβειας, που αναπαριστούν κέρδη. Στο πρόγραμμα όπως θα δούμε παρακάτω αυτά κανονικοποιούνται μεταξύ των τιμών 0 και 1.
  Στον κάθε πίνακα οι γραμμές αναπαριστούν τις στρατηγικές που μπορεί να παίξει ο παίκτης γραμμής, άρα έχουμε m γραμμές
  και οι στήλες αναπαριστούν τις στρατηγικές που μπορεί να παίξει ο παίκτης στήλης, άρα έχουμε n στήλες.
  Αν ο παίκτης γραμμής παίξει την στρατηγική **i** και ο παίκτης στήλης την στρατηγική **j** τότε ο παίκτης γραμμής έχει κέρδος που φαίνεται στον πίνακα **R** στην i γραμμή και j στήλη.
  Αντίστοιχα ο παίκτης στήλης έχει κέρδος που φαίνεται στον πίνακα **C** στην i γραμμή και j στήλη.

- Σημεία έναρξης x_0, y_0 του αλγορίθμου
  Το **x** είναι ένα διάνυσμα πιθανοτήτων (το άθροισμα όλων των στοιχείων του είναι ίσο με 1) μεγέθους **m x 1** και εκφράζει την πιθανότητα με την οποία ο παίκτης γραμμής θα παίξει τις στρατηγικές του. Ουσιαστικά είναι ένα διάνυσμα πιθανοτικών κατανομών πάνω στις στρατηγικές του παίκτη γραμμής.

  Ομοίως το διάνυσμα **y** είναι ένα διάνυσμα πιθανοτήτων (το άθροισμα όλων των στοιχείων του είναι ίσο με 1) μεγέθους **n x 1** και εκφράζει την πιθανότητα με την οποία ο παίκτης στήλης θα παίξει τις στρατηγικές του. Ουσιαστικά είναι ένα διάνυσμα πιθανοτικών κατανομών πάνω στις στρατηγικές του παίκτη στήλης.

  Όπως θα δούμε και πιο αναλυτικά παρακάτω ο αλγόριθμός μας ξεκινάει πάντα από ένα αρχικό σημείο **(x_0,y_0)** και στη συνέχεια εκτελεί την descent procedure. Το αρχικό αυτό σημείο (x_0,y_0) δίνεται στην αρχή του προγράμματος από τον χρήστη, διότι είναι ένα στοιχείο που έχει να κάνει με την συμπεριφορά της εκτέλεσης του αλγορίθμου. Με άλλα λόγια αν από ένα αρχικό σημείο (x_0,y_0) φτάσουμε σε μια ε-κατά προσέγγιση ισορροπία Nash, υπάρχει πιθανότητα από ένα διαφορετικό σημείο $(x`_0,y`_0)$ να φτάσουμε σε μια διαφορετική κατά προσέγγιση ισορροπία Nash.

Τα σημεία **x**, **y** εκφράζουν σημεία πάνω σε ένα αρκετά ιδιόμορφο πολύεδρο.
Στο πρόγραμμά μας θα χρησιμοποιήσουμε ως αρχικά σημεία (x_0,y_0) αυτά που τα στοιχεία τους έχουν ομοιόμορφη κατανομή. Δηλαδή τα στοιχεία του x είναι όλα ίσα με 1/m και τα στοιχεία του y είναι όλα ίσα με 1/n.
Φυσικά μπορούμε να δοκιμάσουμε οποιαδήποτε άλλη κατανομή πιθανοτήτων θέλουμε, αρκεί το άθροισμα των στοιχείων του x (και του y αντίστοιχα) να ισούται με 1.
Για παράδειγμα θα δοκιμάσουμε να ξεκινήσουμε από κάποια γωνία του πολυέδρου, όπου όλα τα στοιχεία του διανύσματος x (αντίστοιχα του y) είναι ίσα με 0, εκτός από ένα που ισούται με 1. Δηλαδή από κάποια αγνή στρατηγική των παικτών.

- Ακρίβεια δ:
  Στην αρχή του προγράμματος καθορίζουμε την ακρίβεια των υπολογισμών μέσω της παραμέτρου **δ** [στο πρόγραμμα έχει την ονομασία **delta**].
  Το δ είναι ένας μικρός αριθμός που στην περίπτωσή μας τον ορίζουμε ίσο με 10^{-3}.
  Έτσι όλοι οι υπολογισμοί και οι έλεγχοι που θα κάνουμε στο πρόγραμμα ακόμα κι αν γίνονται με double ακρίβεια μέσα στο πρόγραμμα, στην πραγματικότητα θα γίνονται με ακρίβεια που ορίζεται από το δ. Αυτό σημαίνει ότι αν το δ έχει οριστεί ίσο με 0.001 τότε οποιοδήποτε νούμερο κάτω του δ (π.χ. 0.0008) θεωρείται ίσο με 0.
  Έτσι ακόμα και στις συγκρίσεις μεταξύ αριθμών δεν θα ελέγχουμε αν είναι ίσοι, αλλά αν η διαφορά τους είναι μικρότερη του δ.

  Το γιατί χρησιμοποιούμε την παράμετρο δ είναι εμφανές. Χρησιμοποιείται για την αποφυγή σφαλμάτων στρογγυλοποίησης (round off errors), καθώς οι υπολογισμοί και οι πράξεις στο συνολικό πρόγραμμα είναι πάρα πολλές και όταν αυτοί γίνονται σε double αριθμούς όπως ορίσαμε στο πρόγραμμά μας τότε επιφέρουν σημαντικά σφάλματα που ενδεχομένως σιγά-σιγά να συσσωρεύονται. Έτσι οι τιμές των μεταβλητών που προκύπτουν μετά από πολλούς υπολογισμούς μπορεί να εμπεριέχουν σημαντικά σφάλματα, που να αλλάζουν και να παραμορφώνουν το τελικό αποτέλεσμα σε πολύ μεγάλο βαθμό. Με το δ ίσο με 10^{-3} θεωρούμε ότι σφάλματα της τάξης του 10^{-4} και μικρότερης τάξης, δεν τα λαμβάνουμε υπόψη.

5.2.2 Τα στοιχεία που παίρνουμε ως Έξοδο στο πρόγραμμα

Το πρόγραμμά μας και ο αλγόριθμος πάνω στον οποίο δουλεύει αυτό, τελικά πρέπει να επιστρέφει την τιμή της κατά προσέγγιση ισορροπίας Nash που υπολόγισε.
Αυτό σημαίνει ότι ως έξοδο στο πρόγραμμα παίρνουμε την τελική τιμή της συνάρτηση **f(x,y)** (Function Evaluation) που υπολογίσαμε, καθώς αυτή σηματοδοτεί την τιμή της κατά προσέγγιση ισορροπίας Nash.
Επίσης ως έξοδο παίρνουμε και το σημείο **(x,y)** στο οποίο αυτήν υπολογίστηκε. Δηλαδή το stationary point στο οποίο σταμάτησε ο αλγόριθμος την descent διαδικασία

5.2.3 Τα ενδιάμεσα αποτελέσματα του προγράμματος

Στο πρόγραμμα παίρνουμε ένα πλήθος ενδιάμεσων αποτελεσμάτων που είναι αρκετά χρήσιμα όχι μόνο ως σημεία ελέγχου της ορθότητας του προγράμματος, αλλά και ως ανεξάρτητα αποτελέσματα που βοηθούν στην κατανόηση της συμπεριφοράς του αλγορίθμου, αλλά και στην εξαγωγή σημαντικών συμπερασμάτων για το πρόβλημα που καλούμαστε να λύσουμε, όπως θα δούμε στο επόμενο Κεφάλαιο που έχει να κάνει με τα πειράματα.

Ως ενδιάμεσα αποτελέσματα θεωρούμε τις πληροφορίες που καταχωρούμε στα διάφορα **info αρχεία**, τα οποία θα παρουσιάσουμε αναλυτικά παρακάτω.

5.3 Βιβλιοθήκες που χρησιμοποιήθηκαν

Η υλοποίηση του αλγορίθμου έγινε σε ANSI C, με την βοήθεια του προγράμματος βελτιστοποίησης CPLEX.
Παρακάτω θα δούμε τις διάφορες βιβλιοθήκες που χρησιμοποιήθηκαν ώστε να μπορέσουμε να χρησιμοποιήσουμε συναρτήσεις και εντολές που αυτές περιέχουν και χρειάστηκαν στο πρόγραμμα.

5.3.1 Τρόπος εισαγωγής μιας βιβλιοθήκης

Στην αρχή του προγράμματος δηλώνουμε με την εντολή include <> τις βιβλιοθήκες που πρόκειται να χρησιμοποιήσουμε στο πρόγραμμα μας. Έτσι μπορούμε να χρησιμοποιούμε συναρτήσεις των βιβλιοθηκών χωρίς να χρειάζεται να τις υλοποιούμε εξ αρχής.

5.3.2 Πρότυπες βιβλιοθήκες της γλώσσας ANSI C

Μια πρότυπη βιβλιοθήκη δεν αποτελεί μέρος της γλώσσας αυτής καθαυτής, όμως ένα περιβάλλον που υποστηρίζει την πρότυπη γλώσσα C, διαθέτει τις δηλώσεις συναρτήσεων αυτής της βιβλιοθήκης.

Οι βιβλιοθήκες που θα χρησιμοποιήσουμε είναι οι εξής:
- Είσοδος και Έξοδος: **<stdio.h>**
- Βοηθητικές Συναρτήσεις **<stdlib.h>**
- Συναρτήσεις Αλφαριθμητικών **<string.h>**
- Μαθηματικές Συναρτήσεις **<math.h>**

Για περαιτέρω ανάλυση των περιεχομένων αυτών των βιβλιοθηκών ανατρέξτε στο [29].

5.3.3 Βιβλιοθήκες που δεν εμπεριέχονται στην ANSI C

Το πρόγραμμα και ο αλγόριθμός μας στηρίζεται πάνω στην επίλυση Γραμμικών Προγραμμάτων (LPs). Για την επίλυση αυτών χρησιμοποιήθηκε το εργαλείο CPLEX το οποίο είναι ένα πρόγραμμα βελτιστοποίησης (optimization software package).
Έτσι εκτός των πρότυπων βιβλιοθηκών, ορίζουμε και μια εξωτερική βιβλιοθήκη βάσει της οποίας θα μπορέσουμε όπως θα δούμε παρακάτω να χρησιμοποιήσουμε πολλές συναρτήσεις για τον ορισμό και την επίλυση γραμμικών προγραμμάτων.
Η βιβλιοθήκη έχει όνομα:
- **ilcplex/cplex.h**

5.3.4 Μια σύντομη παρουσίαση της CPLEX:

Η **CPLEX** είναι ένα πακέτο λογισμικού για βελτιστοποίηση (optimization) προβλημάτων. Στην περίπτωση μας θα χρησιμέψει για να επιλύσουμε τα γραμμικά προγράμματα τα οποία υπάρχουν και αναφέρονται στον αλγόριθμό μας.

Επίσης πρέπει να τονίσουμε ότι το εργαλείο αυτό θα το χρησιμοποιήσουμε ως μαύρο κουτί. Δηλαδή δεν μας ενδιαφέρει το πώς βρίσκει την λύση (η εσωτερική διαδικασία επίλυσης), αλλά το ποια είναι αυτήν βάσει των δεδομένων που θα δίνουμε ως είσοδο και την μέθοδο επίλυσης που θα επιλέξουμε.

Για αυτόν τον λόγο θα χρησιμοποιηθεί η βιβλιοθήκη **cplex.h** η οποία βοηθάει στο να μπορέσεις να χρησιμοποιήσεις την CPLEX μέσω κώδικα C.

Σημείωση: Εκτενέστερη περιγραφή τόσο της λειτουργίας CPLEX, όσο και της χρήσης της βιβλιοθήκης ilcplex/cplex.h, θα γίνει στην συνέχεια.

5.4 Περιγραφή γενικών στοιχείων του κώδικα υλοποίησης

5.4.1 Ορισμός πινάκων

Σε όλο το πρόγραμμα χρησιμοποιούνται πίνακες (είτε διανύσματα) διάφορων μεγεθών.
Αυτοί δημιουργούνται δυναμικά με την χρήση της εντολής **malloc** ώστε να δεσμεύεται τελικά όσος χώρος ακριβώς χρειάζεται στο πρόγραμμα. Όταν οι πίνακες δεν χρειάζονται πια τότε τους ελευθερώνουμε από την εντολή **free**.

Το πρόγραμμα επικεντρώθηκε στην καλή χρήση του χώρου. Δηλαδή προτιμούμαι να αποδεσμεύουμε τον χώρο που δεσμεύεται για κάποιον πίνακα, που ενδεχομένως να χρησιμοποιηθεί αργότερα από το πρόγραμμα. Αυτό έχει σαν αποτέλεσμα να ξαναϋπολογίζουμε τον ίδιο πίνακα πάνω από μία φορές.
Αυτό βέβαια σημαίνει ότι ο χρόνος εκτέλεσης του προγράμματος αυξάνεται, αλλά και υπάρχει μεγαλύτερη πιθανότητα εμφάνισης σφαλμάτων. Και αυτό αποτελεί την αρνητική πλευρά της τακτικής που ακολουθήθηκε.
Όμως παρατηρήθηκε μετά από πολλές εκτελέσεις του προγράμματος, ότι ο χρόνος εκτέλεσης του είναι άκρως ικανοποιητικός [θα παρουσιαστούν σε επόμενο κεφάλαιο].
Επίσης τα σφάλματα καταφέρνουμε να τα ελέγξουμε μέσω της παραμέτρου δ και δεν αποτελούν πρόβλημα της τακτικής υλοποίησης.

5.4.2 Global Μεταβλητές

Στο πρόγραμμά μας όλες οι συναρτήσεις που χρησιμοποιούμε επιστρέφουν την επιθυμητή τιμή της μεταβλητής που θέλουμε στο κεντρικό πρόγραμμα (στην main).
Έτσι δεν χρειάστηκε να χρησιμοποιήσουμε πολλές global μεταβλητές.
Οι global μεταβλητές που χρησιμοποιούμε είναι δύο κατηγοριών.

5.4.2.1 Απλές Μεταβλητές

Είναι αυτές που χρησιμοποιούμε ώστε να βοηθηθούμε σε υπολογισμούς, ελέγχους ή στην δημιουργία αρχείων πληροφοριών μέσα στην main, όπως το πλήθος των στοιχείων του συνόλου $S_R(y)$ [index_R_y_c] και του $S_C(x)$ [index_Ct_x_c] που

υπολογίζονται μέσα στην συνάρτηση populatebyrow_3, αλλά είναι απαραίτητα στην main ώστε να γίνουν κάποιοι υπολογισμοί και στο να δημιουργηθεί το βασικό αρχείο πληροφοριών του προγράμματος info.txt που θα αναλυθεί παρακάτω.
Τον ίδιο σκοπό εξυπηρετούν οι global μεταβλητές H_global και f_val_4_b που εκφράζουν τις τιμές που παίρνουν οι μεταβλητές H και η τιμή της συνάρτησης f(x,y) μέσα σε μία από τις συναρτήσεις (συγκεκριμένα στην epsilon_double_star).

5.4.2.2 Αρχεία εισόδου και εξόδου

Ως global μεταβλητές ορίζουμε και τους δείκτες αρχείων που θα χρησιμοποιηθούν για την δημιουργία αρχείων είτε εξόδου είτε εισόδου πληροφοριών και δεδομένων του προγράμματος. Αυτό γίνεται μέσω της βιβλιοθήκης **<stdio.h>**

Έτσι έχουμε τα παρακάτω global αρχεία:

Αρχεία εξόδου:

- **info.txt:**
  Το αρχείο αυτό είναι αρχείο κειμένου και περιέχει τις απαραίτητες πληροφορίες εξόδου που χρειάζεται να ξέρουμε μετά το πέρας της εκτέλεσης του αλγορίθμου. Τα στοιχεία που υπάρχουν στο αρχείο και η μορφή τους θα αναλυθούν παρακάτω.

  Πρέπει να πούμε ότι τα δεδομένα παρουσιάζονται σε στήλες στην σειρά.
  Κάθε στήλη έχει ένα τίτλο και κάθε επανάληψη (loop) του αλγορίθμου βρίσκεται σε διαφορετική γραμμή.

  Αρχικά υπάρχουν πληροφορίες κάτω από τον τίτλο `Info:` για το τι σημαίνουν οι συμβολισμοί των τίτλων των στηλών που παρουσιάζονται παρακάτω στο αρχείο.
  Στη συνέχεια παρουσιάζονται πληροφορίες για το **μέγεθος του προβλήματος**.
  Ουσιαστικά καταγράφεται το μέγεθος των πινάκων R και C [οι τιμές των m και n δηλαδή].
  Επίσης καταγράφεται και η τιμή του **δ** (delta).

  Στη συνέχεια καταγράφονται δεδομένα που δημιουργούνται κατά την διάρκεια της εκτέλεσης του προγράμματος σε κάθε επανάληψη του αλγορίθμου, τόσο για το primal πρόβλημα, όσο και για το dual πρόβλημα όπως θα δούμε και στην ανάλυση του αλγορίθμου.

  Έτσι λοιπόν έχουμε 21 στήλες δεδομένων που εξάγει το πρόγραμμα.

  Ας τα δούμε αναλυτικά:
  - **R** είναι ο αύξων αριθμός της επανάληψης του αλγορίθμου [του κεντρικού loop]. Η αρίθμηση αρχίζει από το 0 και αυξάνεται κάθε φορά που εκτελείται από την αρχή ο αλγόριθμος.

  - **f_rb** είναι η τιμή που παίρνει η συνάρτηση $f_R(x,y)$ στην αρχή της εκτέλεσης του αλγορίθμου, πριν εκτελεστεί το βήμα 1

  - **f_cb** είναι η τιμή που παίρνει η συνάρτηση $f_C(x,y)$ στην αρχή της εκτέλεσης του αλγορίθμου, πριν εκτελεστεί το βήμα 1

- **f1b** είναι η τιμή που παίρνει η συνάρτηση f(x,y) στην αρχή της εκτέλεσης του αλγορίθμου, πριν εκτελεστεί το βήμα 1

- **f_ra** είναι η τιμή που παίρνει η συνάρτηση $f_R(x,y)$ αφού εκτελεστεί το βήμα 1

- **f_ca** είναι η τιμή που παίρνει η συνάρτηση $f_C(x,y)$ αφού εκτελεστεί το βήμα 1

- **f1a** είναι η τιμή που παίρνει η συνάρτηση f(x,y) αφού εκτελεστεί το βήμα 1

- **V(x,y)** είναι η τιμή της αντικειμενικής συνάρτησης που παίρνουμε από την λύση του minimax προβλήματος στο σημείο (x,y) κατά την εκτέλεση του δεύτερου βήματος του αλγορίθμου.

- **ρ** είναι η τιμή του ρ που υπολογίζεται στο δεύτερο βήμα του αλγορίθμου

- **1-ρ** είναι η τιμή του 1-ρ που υπολογίζεται στο δεύτερο βήμα του αλγορίθμου.

- **L** είναι η τιμή του λ όπως αυτό υπολογίζεται μετά το δεύτερο βήμα του αλγορίθμου

- **Mi** είναι η τιμή του μ όπως αυτό υπολογίζεται μετά το δεύτερο βήμα του αλγορίθμου

- **epsilon*** είναι η τιμή που παίρνει το ε για τον υπολογισμό της f(x,y) στο βήμα 4, όπως περιγράφεται στο Appendix A.3 του paper [1].

- **H** είναι η τιμή του H που υπολογίζεται ώστε να βρούμε την βελτιωμένη τιμή του ε [ε**] μέσα στην συνάρτηση epsilon_double_star

- **epsilon**** είναι η βελτιωμένη τιμή του ε για το βήμα 4 του αλγορίθμου. Θα δούμε παρακάτω πως υπολογίζονται όλα αυτά.

- **S_R** είναι το πλήθος των στοιχείων του συνόλου support $S_R(y)$ όπως αυτό διαμορφώνεται μετά το βήμα 4 [στο πρόγραμμα αναφέρεται ως index_R_y_c]

- **S_C** είναι το πλήθος των στοιχείων του συνόλου support $S_C(x)$ όπως αυτό διαμορφώνεται μετά το βήμα 4 [στο πρόγραμμα αναφέρεται ως index_Ct_x_c]

- **f_R_4** είναι η τιμή που παίρνει η συνάρτηση $f_R(x,y)$ στο τέλος της εκτέλεσης του αλγορίθμου, αφού εκτελεστεί το βήμα 4

- **f_C_4** είναι η τιμή που παίρνει η συνάρτηση $f_C(x,y)$ στο τέλος της εκτέλεσης του αλγορίθμου, αφού εκτελεστεί το βήμα 4

- **f4** είναι η τιμή που παίρνει η συνάρτηση f(x,y) στο τέλος της εκτέλεσης του αλγορίθμου, αφού εκτελεστεί το βήμα 4

- **f4_formula** είναι η τιμή που παίρνει η συνάρτηση f(x,y) στο τέλος της εκτέλεσης του αλγορίθμου, αφού εκτελεστεί το βήμα 4 βάσει μιας φόρμουλας που

θα δώσουμε παρακάτω. Υπολογίστηκε ώστε να επαληθευτεί η τιμή της f4 παραπάνω.

Έτσι η τιμή της κατά προσέγγιση ισορροπίας Nash που βρίσκουμε είναι η τιμή της **f4** στην τελευταία επανάληψη του αλγορίθμου.
Το σημείο **(x,y)** στο οποίο έχουμε βρει ισορροπία Nash φαίνεται ακριβώς μετά, καθώς και οι τιμές των $\mathbf{x^T*R*y}$ και $\mathbf{x^T*C*y}$ (xt_R_y και xt_C_y αντίστοιχα στο αρχείο) στο τέλος των επαναλήψεων του αλγορίθμου.

Αφού αναπαρασταθεί ακριβώς με τον ίδιο τρόπο η dual εκτέλεση του αλγορίθμου, στο τέλος έχουμε την ισορροπία Nash που έχουμε βρει τελικά. Είναι η μικρότερη μεταξύ των τελικών f4 που βρήκαμε στην επίλυση του primal και του dual προβλήματος.

- **info_x_y_1a.txt:**
  Φαίνονται οι τιμές των x και y όλων των βημάτων πριν το βήμα 1.

- **info_x_y_1b.txt:**
  Φαίνονται οι τιμές των x και y όλων των βημάτων μετά το βήμα 1.

- **info_x_y_2.txt:**
  Φαίνονται οι τιμές των x και y όλων των βημάτων μετά το βήμα 2.

- **info_w_z.txt:**
  Φαίνονται οι (dual) τιμές των w και z όλων των βημάτων μετά το βήμα 2.

- **info_x_y_4.txt:**
  Φαίνονται οι τιμές των x και y όλων των βημάτων μετά το βήμα 4.

- **steps.txt:**
  Φαίνονται το πώς εκτελέστηκαν τα βήματα σε κάθε βήμα του αλγορίθμου μέχρι το τέλος του. Χρησιμοποιήθηκε για έλεγχο των goto του προγράμματος.

- **r_c_m_p.txt:**
  Φαίνονται εκτυπωμένα οι πίνακες R και C αντίστοιχα, στην κανονική τους μορφή [m x n μεγέθους]

<u>Αρχεία εισόδου:</u>

- **r_c_m.txt:**
  Το αρχείο αυτό χρησιμοποιείται ως αρχείο εισόδου, ώστε να εισάγουμε τα στοιχεία των πινάκων R και C του προβλήματος.

  <u>Ο τρόπος εισαγωγής των πινάκων R και C είναι ο εξής:</u>
  Πρώτα εισάγουμε τα στοιχεία του πίνακα R και συνέχεια του πίνακα C.
  Τα στοιχεία όλα πρέπει να είναι στην ίδια γραμμή το ένα μετά το άλλο χωρισμένα με tab το ένα με το άλλο.
  Στην αρχή μπαίνουν τα στοιχεία της πρώτης γραμμής του πίνακα R, στη συνέχεια

της δεύτερης γραμμής κοκ μέχρι να τελειώσει ο πίνακας R.
Έπειτα συνεχίζουμε με τον πίνακα C ομοίως.

Σημείωση: Τα στοιχεία είναι double ακρίβειας.

5.5 Περιγραφή Συναρτήσεων

5.5.1 Συναρτήσεις για πράξεις σε πινάκες

Παρακάτω περιγράφονται οι συναρτήσεις που δημιουργήθηκαν έτσι ώστε να μπορέσουμε να κάνουμε κάποιες πράξεις πάνω σε πίνακες και διανύσματα. Επίσης αναφέρουμε ξανά ότι τα στοιχεία όλων των πινάκων που χρησιμοποιούμαι είναι ακρίβειας double.

void print_matrix(double **my_matrix, int row_size, int col_size)
Η συνάρτηση αυτή χρησιμοποιείται για να εκτυπώσει στην οθόνη οποιοδήποτε μεγέθους πίνακα.

Παίρνει ως πρώτο όρισμα τον πίνακα my_matrix που θέλουμε να εκτυπώσουμε στην οθόνη και έχει στοιχεία δεκαδικούς double ακρίβειας, ως δεύτερο όρισμα τον αριθμό των γραμμών του πίνακα (row_size) και ως τρίτο όρισμα τον αριθμό των στηλών του πίνακα (col_size).

Η συνάρτηση δεν επιστρέφει τίποτα, απλά εκτυπώνει στην οθόνη τον πίνακα που βάλαμε ως πρώτο όρισμα.

double ** normilize(double **my_matrix, int row_size, int col_size);
Όπως αναφέραμε παραπάνω χωρίς βλάβη της γενικότητας σε κάθε παίγνιο μπορούμε να κανονικοποιήσουμε τις τιμές των πινάκων κέρδους του κάθε παίχτη μεταξύ των αριθμών 0 και 1.
Για αυτόν τον λόγο η συνάρτηση αυτή χρησιμοποιείται για να κανονικοποιεί τις τιμές ενός πίνακα, μεταξύ των αριθμών 0 και 1.

Παίρνει ως πρώτο όρισμα τον πίνακα my_matrix που θέλουμε να κανονικοποιήσουμε, ως δεύτερο όρισμα τον αριθμό των γραμμών του πίνακα (row_size) και ως τρίτο όρισμα τον αριθμό των στηλών του πίνακα (col_size).

Η διαδικασία κανονικοποίησης για οποιονδήποτε πίνακα (ας υποθέσουμε τον **R**) είναι η εξής:
Αρχικά βρίσκουμε το μέγιστο στοιχείο του πίνακα R μέσω της συνάρτησης find_max που θα δούμε παρακάτω και το αποθηκεύουμε στην μεταβλητή max, καθώς επίσης και το ελάχιστο στοιχείο του πίνακα R μέσω της συνάρτησης find_min που θα δούμε παρακάτω και το αποθηκεύουμε στην μεταβλητή min.
Στη συνέχεια για κάθε στοιχείο του πίνακα κάνουμε την κανονικοποίηση ως εξής:
Έστω ότι βρισκόμαστε στο σημείο **(i,j)**. Τότε το στοιχείο R(i,j) που βρίσκεται σε εκείνη την θέση παίρνει νέα τιμή που βρίσκεται με τον εξής τρόπο:

$$R_{ij} = \frac{R_{ij} - \min\limits_{(i,j)}\{R_{ij}\}}{\max\limits_{(i,j)}\{R_{ij}\} - \min\limits_{(i,j)}\{R_{ij}\}} \qquad \text{για κάθε i = 1…m και j = 1…n}$$

Με αυτόν τον τρόπο κάνουμε shifting και scaling των στοιχείων του πίνακα ώστε όλες οι τιμές του να είναι μεταξύ 0 και 1.

ΠΡΟΣΟΧΗ: Η παραπάνω κανονικοποίηση δεν δουλεύει μόνο όταν το max είναι ίσο με το min του πίνακα, γιατί ο παρονομαστής γίνεται ίσος με 0. Αλλά στην περίπτωσή μας δεν μας ενδιαφέρουν τέτοιου είδους προβλήματα γιατί η λύση τους είναι τετριμμένη.

*[Ομοίως πράττουμε και για τον πίνακα κέρδους του παίκτη στήλης **C**.]*

Η συνάρτηση επιστρέφει τον νέο κανονικοποιημένο πίνακα my_matrix.

double **tran_matrix(double **my_matrix, int row_size, int col_size)
Η συνάρτηση αυτή βρίσκει τον ανάστροφο ενός πίνακα.

Παίρνει ως πρώτο όρισμα τον πίνακα my_matrix που θέλουμε να αναστρέψουμε, ως δεύτερο όρισμα τον αριθμό των γραμμών του πίνακα (row_size) και ως τρίτο όρισμα τον αριθμό των στηλών του πίνακα (col_size).

Μέσα στη συνάρτηση απλά παίρνουμε κάθε ένα στοιχείο του πίνακα my_matrix που βρίσκεται στην θέση **(i,j)** και το δημιουργούμε έναν νέο πίνακα new_my_matrix που έχει το στοιχείο αυτό στην θέση **(j,i).**
Με αυτόν τον τρόπο δημιουργούμε νέο πίνακα που είναι η αναστροφή του προηγούμενου. Δηλαδή οι γραμμές του αρχικού έγιναν στήλες στον νέο.

Η συνάρτηση επιστρέφει τον νέο αντεστραμμένο πίνακα new_my_matrix.

double ** norm_prob_vector(double **vector, int row_size)
Η συνάρτηση αυτή κανονικοποιεί τα διανύσματα πιθανοτήτων.

Παίρνει ως πρώτο όρισμα το διάνυσμα πιθανοτήτων vector μεγέθους row_size x 1 που θέλουμε να κανονικοποιήσουμε και ως δεύτερο όρισμα τον αριθμό των γραμμών του διανύσματος (row_size).

Χρησιμοποιούμε αυτήν την συνάρτηση ώστε να μειώσουμε τα σφάλματα που δημιουργούνται στα διανύσματα πιθανοτήτων λόγω των πράξεων μεταξύ double στο πρόγραμμα και έχουν ως αποτέλεσμα μη ορθά αποτελέσματα λόγω μεταβίβασης σφαλμάτων στρογγυλοποίησης (round off errors) στο πρόγραμμα και ως τελικό αποτέλεσμα συσσώρευση σφαλμάτων και εξαγωγή λανθασμένων αποτελεσμάτων.

Με την συνάρτηση αυτήν απλώς κανονικοποιούμε το διάνυσμα πιθανοτήτων ώστε τελικά όλα του τα στοιχεία να αθροίζουν στο 1.
Αρχικά όσα στοιχεία είναι αρνητικά τα κάνουμε ίσα με 0. Αυτό γιατί υπάρχει το ενδεχόμενο να έχουμε αρνητικές πιθανότητες μετά από τόσους υπολογισμούς. Βέβαια

αυτές θα είναι αρκετά μικρής τάξης, αλλά ενδέχεται να δημιουργήσουν πρόβλημα στην εκτέλεση του προγράμματος.
Στη συνέχεια βρίσκουμε το άθροισμα (sum) όλων των στοιχείων του πίνακα πιθανοτήτων. Εν συνεχεία απλώς διαιρούμε κάθε στοιχείο του πίνακα με το συνολικό άθροισμα sum.
Με αυτό το scaling που κάνουμε, δημιουργείται ένα νέο διάνυσμα vector που έχει όλα του τα στοιχεία ίσα με 1 και εκφράζει τις ίδιες πιθανότητες όπως και πριν.

Η συνάρτηση επιστρέφει το νέο κανονικοποιημένο διάνυσμα πιθανοτήτων vector.

double **v_min_v(double **vector1, double **vector2, int row_size)
Η συνάρτηση αυτή εκτελεί την πράξη της αφαίρεσης μεταξύ δύο διανυσμάτων μεγέθους row_size x 1.

Παίρνει ως πρώτο όρισμα το διάνυσμα vector1 μεγέθους row_size x 1, ως δεύτερο όρισμα το διάνυσμα vector2 μεγέθους row_size x 1 που θέλουμε να αφαιρέσουμε από το πρώτο διάνυσμα και ως τρίτο όρισμα παίρνει τον αριθμό των γραμμών του διανύσματος (row_size) τόσο του vector1 όσο και του vector2 [για να τα αφαιρέσουμε πρέπει να έχουν το ίδιο μέγεθος].

Μέσα στην συνάρτηση απλώς αφαιρούμε κάθε στοιχείο που βρίσκεται στη θέση (i,j) του διανύσματος vector2 από το αντίστοιχο στοιχείο που βρίσκεται στη θέση (i,j) στο διάνυσμα vector1.

Η συνάρτηση επιστρέφει το νέο διάνυσμα my_new_vector που προέκυψε από την αφαίρεση των δύο διανυσμάτων.

double **v_sum_v(double **vector1, double **vector2, int row_size)
Η συνάρτηση αυτή εκτελεί την πράξη της πρόσθεσης μεταξύ δύο διανυσμάτων μεγέθους row_size x 1.

Παίρνει ως πρώτο όρισμα το διάνυσμα vector1 μεγέθους row_size x 1, ως δεύτερο όρισμα το διάνυσμα vector2 μεγέθους row_size x 1 που θέλουμε να προσθέσουμε στο το πρώτο διάνυσμα και ως τρίτο όρισμα παίρνει τον αριθμό των γραμμών του διανύσματος (row_size) τόσο του vector1 όσο και του vector2 [για να τα προσθέσουμε πρέπει να έχουν το ίδιο μέγεθος].

Μέσα στην συνάρτηση απλώς προσθέτουμε κάθε στοιχείο που βρίσκεται στη θέση (i,j) του διανύσματος vector1 με το αντίστοιχο στοιχείο που βρίσκεται στη θέση (i,j) στο διάνυσμα vector2.

Η συνάρτηση επιστρέφει το νέο διάνυσμα my_new_vector που προέκυψε από την πρόσθεση των δύο διανυσμάτων.

double **v_m_mul(double **my_matrix, int row_size, int col_size, double **vector)

Η συνάρτηση αυτή εκτελεί την πράξη του πολλαπλασιασμού μεταξύ ενός διανύσματος μεγέθους row_size x 1 και ενός πίνακα μεγέθους row_size x col_size.

Παίρνει ως πρώτο όρισμα τον πίνακα my_matrix μεγέθους row_size x col_size, ως δεύτερο όρισμα τον αριθμό των γραμμών (row_size) τόσο του πίνακα my_matrix όσο και του διανύσματος vector [για να πολλαπλασιάσουμε διάνυσμα με πίνακα πρέπει να έχουν ίδιο αριθμό γραμμών] και ως τρίτο όρισμα τον αριθμό των στηλών του πίνακα my_matrix. Ως τέταρτο όρισμα παίρνει το διάνυσμα vector μεγέθους row_size x 1.

Το αποτέλεσμα της πράξης πολλαπλασιασμού διανύσματος-πίνακα είναι ένα διάνυσμα μεγέθους 1 x col_size.
Έτσι λοιπόν για να βρούμε για παράδειγμα το (0,i) στοιχείο του νέου διανύσματος πολλαπλασιάζουμε κάθε στοιχείο (0,j) του διανύσματος vector με το (i,j) στοιχείο του πίνακα my_matrix και παίρνουμε το άθροισμα όλων αυτών που προκύπτουν για κάθε j.

Η συνάρτηση επιστρέφει το νέο διάνυσμα my_new_vector που προέκυψε από τον πολλαπλασιασμό μεταξύ του διανύσματος και του πίνακα.

double **m_v_mul(double **my_matrix, int row_size, int col_size, double **y)
Η συνάρτηση αυτή εκτελεί την πράξη του πολλαπλασιασμού μεταξύ ενός πίνακα μεγέθους row_size x col_size και ενός διανύσματος μεγέθους row_size x 1.

Παίρνει ως πρώτο όρισμα τον πίνακα my_matrix μεγέθους row_size x col_size, ως δεύτερο όρισμα τον αριθμό των γραμμών (row_size) τόσο του πίνακα my_matrix όσο και του διανύσματος y [για να πολλαπλασιάσουμε διάνυσμα με πίνακα πρέπει να έχουν ίδιο αριθμό γραμμών] και ως τρίτο όρισμα τον αριθμό των στηλών του πίνακα my_matrix. Ως τέταρτο όρισμα παίρνει το διάνυσμα y μεγέθους row_size x 1.

Το αποτέλεσμα της πράξης πολλαπλασιασμού πίνακα- διανύσματος είναι ένα διάνυσμα μεγέθους row_size x 1.
Έτσι λοιπόν για να βρούμε για παράδειγμα το (i,0) στοιχείο του νέου διανύσματος πολλαπλασιάζουμε κάθε στοιχείο (j,0) του διανύσματος y με το (i,j) στοιχείο του πίνακα my_matrix και παίρνουμε το άθροισμα όλων αυτών που προκύπτουν για κάθε j.

Η συνάρτηση επιστρέφει το νέο διάνυσμα my_new_vector που προέκυψε από τον πολλαπλασιασμό μεταξύ του πίνακα και του διανύσματος.

double **v_e_mul(double **vector, int col_size, double element)
Η συνάρτηση αυτή εκτελεί την πράξη του πολλαπλασιασμού μεταξύ ενός διανύσματος μεγέθους 1 x col_size και ενός στοιχείου μεγέθους 1x1.

Παίρνει ως πρώτο όρισμα το διάνυσμα vector μεγέθους 1 x col_size, ως δεύτερο όρισμα τον αριθμό των στηλών (col_size) του διανύσματος και ως τρίτο όρισμα το στοιχείο με το οποίο θέλουμε να πολλαπλασιάσουμε όλα τα στοιχεία του διανύσματος.

Το αποτέλεσμα της πράξης πολλαπλασιασμού διανύσματος-στοιχείου είναι ένα διάνυσμα μεγέθους 1 x col_size.
Έτσι λοιπόν για να βρούμε το νέο διάνυσμα απλώς πολλαπλασιάζουμε κάθε ένα στοιχείο του (0,i) με το στοιχείο element.

Η συνάρτηση επιστρέφει το νέο διάνυσμα new_my_matrix που προέκυψε από τον πολλαπλασιασμό μεταξύ του διανύσματος και του στοιχείου.

double **e_v_mul(double **vector, int row_size, double element)
Η συνάρτηση αυτή εκτελεί την πράξη του πολλαπλασιασμού μεταξύ ενός στοιχείου μεγέθους 1x1 και ενός διανύσματος μεγέθους row_size x 1.

Παίρνει ως πρώτο όρισμα το διάνυσμα vector μεγέθους row_size x 1, ως δεύτερο όρισμα τον αριθμό των γραμμών (row_size) του διανύσματος και ως τρίτο όρισμα το στοιχείο με το οποίο θέλουμε να πολλαπλασιάσουμε όλα τα στοιχεία του διανύσματος.

Το αποτέλεσμα της πράξης πολλαπλασιασμού στοιχείου-διανύσματος στην συγκεκριμένη περίπτωση είναι ένα διάνυσμα μεγέθους row_size x 1.
Έτσι λοιπόν για να βρούμε το νέο διάνυσμα απλώς πολλαπλασιάζουμε κάθε ένα στοιχείο του (i,0) με το στοιχείο element.

Η συνάρτηση επιστρέφει το νέο διάνυσμα new_my_matrix που προέκυψε από τον πολλαπλασιασμό μεταξύ του στοιχείου και του διανύσματος.

double **v_v_mul(double **vector_1, int row_size, double **vector_2, int col_size)
Η συνάρτηση αυτή εκτελεί την πράξη του πολλαπλασιασμού μεταξύ ενός διανύσματος μεγέθους row_size x 1 και ενός διανύσματος μεγέθους 1 x col_size.

Παίρνει ως πρώτο όρισμα το διάνυσμα vector_1 μεγέθους row_size x 1 και ως δεύτερο όρισμα τον αριθμό των γραμμών (row_size) του διανύσματος vector_1.
Ως τρίτο όρισμα παίρνει το διάνυσμα vector_2 μεγέθους 1 x col_size και ως τέταρτο όρισμα τον αριθμό των στηλών (col_size) του διανύσματος vector_2.

Το αποτέλεσμα της πράξης αυτού του πολλαπλασιασμού μεταξύ διανύσματος - διανύσματος είναι ένας πίνακας μεγέθους row_size x col_size.
Έτσι λοιπόν για να βρούμε για παράδειγμα το (i,j) στοιχείο του νέου πίνακα πολλαπλασιάζουμε κάθε στοιχείο (i,0) του διανύσματος vector_1 με το (0,j) στοιχείο του διανύσματος vector_2 και παίρνουμε το άθροισμα όλων αυτών που προκύπτουν για κάθε i και j.

Η συνάρτηση επιστρέφει τον νέο πίνακα new_my_matrix που προέκυψε από τον πολλαπλασιασμό μεταξύ του πρώτου και του δεύτερου διανύσματος.

double **v_v_mul_2(double **vector_1, int col_size, double **vector_2, int row_size)

Η συνάρτηση αυτή εκτελεί την πράξη του πολλαπλασιασμού μεταξύ ενός διανύσματος μεγέθους 1 x col_size και ενός διανύσματος μεγέθους row_size x 1.

Παίρνει ως πρώτο όρισμα το διάνυσμα vector_1 μεγέθους 1 x col_size και ως δεύτερο όρισμα τον αριθμό των στηλών (col_size) του διανύσματος vector_1.
Ως τρίτο όρισμα παίρνει το διάνυσμα vector_2 μεγέθους row_size x 1 και ως τέταρτο όρισμα τον αριθμό των γραμμών (col_size) του διανύσματος vector_2.

Για να γίνει ο πολλαπλασιασμός αυτός πρέπει το δεύτερο και τέταρτο όρισμα να είναι ίσα.

Το αποτέλεσμα της πράξης αυτού του πολλαπλασιασμού μεταξύ διανύσματος - διανύσματος είναι ένα στοιχείο μεγέθους 1 x 1.
Έτσι λοιπόν για να βρούμε για παράδειγμα το (i,j) στοιχείο του νέου πίνακα πολλαπλασιάζουμε κάθε στοιχείο (0,i) του διανύσματος vector_1 με το (j,0) στοιχείο του διανύσματος vector_2 και παίρνουμε το άθροισμα όλων αυτών που προκύπτουν για κάθε i και j.

Η συνάρτηση επιστρέφει τον νέο πίνακα new_my_matrix μεγέθους 1 x 1 που προέκυψε από τον πολλαπλασιασμό μεταξύ του πρώτου και του δεύτερου διανύσματος.

5.5.2 Συναρτήσεις για την Descent Διαδικασία του αλγορίθμου

double find_max(double **my_matrix, int row_size, int col_size)
Η συνάρτηση αυτή βρίσκει το μέγιστο στοιχείο μεταξύ όλων των στοιχείων ενός πίνακα μεγέθους row_size x col_size.

Παίρνει ως πρώτο όρισμα τον πίνακα my_matrix μεγέθους row_size x col_size, ως δεύτερο όρισμα τον αριθμό των γραμμών (row_size) του πίνακα my_matrix και ως τρίτο όρισμα τον αριθμό των στηλών (col_size) του πίνακα my_matrix.

Στην συνάρτηση διατρέχουμε ένα-ένα τα στοιχεία του πίνακα και με μια απλή σύγκριση κρατάμε κάθε φορά το μεγαλύτερο από αυτά.

Η συνάρτηση επιστρέφει το μεγαλύτερο στοιχείο max του πίνακα.

double find_max_s(double **my_matrix, int row_size, int col_size, int **index, int index_size)
Η συνάρτηση αυτή βρίσκει το μέγιστο στοιχείο μεταξύ όχι όλων των στοιχείων ενός διανύσματος μεγέθους row_size x 1, αλλά ενός υποσυνόλου του πίνακα το οποίο σύνολο περιέχει όποια στοιχεία του διανύσματος μεταξύ των οποίων θέλουμε να βρούμε το μέγιστο και σηματοδοτούνται από ένα διάνυσμα index μεγέθους index_size.

Το διάνυσμα index είναι ένα διάνυσμα που περιέχει δείκτες στα στοιχεία του διανύσματος my_matrix μεταξύ των οποίων θέλουμε να βρούμε το μέγιστο.

Αυτά εμφανίζονται κατά σειρά στο διάνυσμα index και όταν τελειώσει η λίστα με το υποσύνολο τότε στις υπόλοιπες θέσεις περιέχει την τιμή -1.
Δηλαδή αν θέλουμε να βρούμε το μέγιστο μεταξύ του πρώτου, του τρίτου και του τέταρτου στοιχείου ενός διανύσματος που περιέχει πέντε στοιχεία τότε το διάνυσμα index θα είναι της μορφής:

$$\mathbf{index = [0\ \ 2\ \ 3\ \ -1\ \ -1]^T}$$

Παίρνει ως πρώτο όρισμα το διάνυσμα my_matrix μεγέθους row_size x 1, ως δεύτερο όρισμα τον αριθμό των γραμμών (row_size) του πίνακα my_matrix και ως τρίτο όρισμα τον αριθμό των στηλών (col_size) του πίνακα my_matrix [εδώ είναι πάντα ίσο με 1].
Ως τέταρτο όρισμα παίρνει όπως είπαμε το διάνυσμα index με τους δείκτες στα στοιχεία του διανύσματος my_matrix μεταξύ των οποίων θέλουμε το μέγιστο και ως πέμπτο όρισμα παίρνει το πλήθος των στοιχείων που αποτελούν το υποσύνολο (ουσιαστικά το πλήθος των στοιχείων του διανύσματος index που δεν έχουν την τιμή -1).

Στην συνάρτηση διατρέχουμε ένα-ένα τα στοιχεία του διανύσματος my_matrix πλήθους index_size που σηματοδοτούνται από το διάνυσμα index και με μια απλή σύγκριση μεταξύ τους κρατάμε κάθε φορά το μεγαλύτερο από αυτά.

Η συνάρτηση επιστρέφει το μεγαλύτερο στοιχείο του πίνακα max.

double find_min(double **my_matrix, int row_size, int col_size)
Η συνάρτηση αυτή βρίσκει το ελάχιστο στοιχείο μεταξύ όλων των στοιχείων ενός πίνακα μεγέθους row_size x col_size.

Παίρνει ως πρώτο όρισμα τον πίνακα my_matrix μεγέθους row_size x col_size, ως δεύτερο όρισμα τον αριθμό των γραμμών (row_size) του πίνακα my_matrix και ως τρίτο όρισμα τον αριθμό των στηλών (col_size) του πίνακα my_matrix.

Στην συνάρτηση διατρέχουμε ένα-ένα τα στοιχεία του πίνακα και με μια απλή σύγκριση κρατάμε κάθε φορά το ελάχιστο από αυτά.

Η συνάρτηση επιστρέφει το μεγαλύτερο στοιχείο min του πίνακα.

int **suppmax_u(double **vector, int row_size)
Η συνάρτηση αυτή βρίσκει το σύνολο όλων των στοιχείων που είναι ίσα με το μέγιστο στοιχείο ενός διανύσματος μεγέθους row_size x 1.

Παίρνει ως πρώτο όρισμα το διάνυσμα vector μεγέθους row_size x 1 και ως δεύτερο όρισμα τον αριθμό των γραμμών (row_size) του διανύσματος.

Στην συνάρτηση αρχικά μέσω της συνάρτηση find_max βρίσκουμε το μέγιστο από τα στοιχεία του διανύσματος. Διατρέχουμε ένα-ένα όλα τα στοιχεία του διανύσματος vector και ελέγχουμε αν είναι ίσα με την μέγιστη τιμή που είχαμε βρει προηγουμένως. Αν ναι τότε προσθέτουμε τον δείκτη της θέσης αυτής σε ένα νέο

διάνυσμα new_my_matrix μεγέθους row_size x 1. Στις εναπομείναντες θέσεις του new_my_matrix βάζουμε την τιμή -1.
Πιο συγκεκριμένα αν έχουμε μέγιστη τιμή max στο διάνυσμα vector μεγέθους 5 και αυτή η τιμή βρίσκεται στις θέσεις 0, 3 και 4 του διανύσματός vector, τότε το new_my_matrix θα πάρει τελικά τις εξής τιμές:

$$\textbf{new\_my\_matrix} = [0 \;\; 3 \;\; 4 \;\; -1 \;\; -1]^T$$

Η συνάρτηση επιστρέφει το διάνυσμα new_my_matrix.

ΠΡΟΣΟΧΗ: για λόγους ευστάθειας όπως αναπτύξαμε και παραπάνω, χρησιμοποιείται η παράμετρος **δ**. Αντί να αναζητούμε τα στοιχεία του διανύσματος που έχουν τιμή ακριβώς max, αναζητούμε αυτά τα στοιχεία που έχουν τιμή μεγαλύτερη ή ίση του **max-δ**.
Ουσιαστικά σε όλο το πρόγραμμα δουλεύουμε προσεγγιστικά στις double τιμές σύμφωνα πάνα με την παράμετρο δ που μένει σταθερή σε όλο το πρόγραμμα.

int **suppmax_u_bar(double **vector, int row_size)
Η συνάρτηση αυτή δουλεύει όπως ακριβώς η συνάρτηση suppmax_u, απλώς κάνει την αντίθετη διαδικασία. Βρίσκει όλα τα στοιχεία που δεν είναι ίσα με το μέγιστο στοιχείο ενός διανύσματος μεγέθους row_size x 1.

Παίρνει ως πρώτο όρισμα το διάνυσμα vector μεγέθους row_size x 1 και ως δεύτερο όρισμα τον αριθμό των γραμμών (row_size) του διανύσματος.

Στην συνάρτηση αρχικά μέσω της συνάρτηση find_max βρίσκουμε το μέγιστο από τα στοιχεία του διανύσματος. Διατρέχουμε ένα-ένα όλα τα στοιχεία του διανύσματος vector και ελέγχουμε αν δεν είναι ίσα με την μέγιστη τιμή που είχαμε βρει προηγουμένως. Αν δεν είναι τότε προσθέτουμε τον δείκτη της θέσης αυτής σε ένα νέο διάνυσμα new_my_matrix μεγέθους row_size x 1. Στις εναπομείναντες θέσεις του new_my_matrix βάζουμε την τιμή -1.
Πιο συγκεκριμένα αν έχουμε μέγιστη τιμή max στο διάνυσμα vector μεγέθους 5 και αυτή η τιμή βρίσκεται στις θέσεις 0, 3 και 4 του διανύσματός vector, τότε το new_my_matrix θα πάρει τελικά τις εξής τιμές:

$$\textbf{new\_my\_matrix} = [1 \;\; 2 \;\; -1 \;\; -1 \;\; -1]^T$$

Η συνάρτηση επιστρέφει το διάνυσμα new_my_matrix.

ΠΡΟΣΟΧΗ: για λόγους ευστάθειας όπως αναπτύξαμε και παραπάνω, χρησιμοποιείται η παράμετρος **δ**. Αντί να αναζητούμε τα στοιχεία του διανύσματος που δεν έχουν τιμή ακριβώς max, αναζητούμε αυτά τα στοιχεία που έχουν τιμή μικρότερη του **max-δ**.
Ουσιαστικά σε όλο το πρόγραμμα δουλεύουμε προσεγγιστικά στις double τιμές σύμφωνα πάνα με την παράμετρο δ που μένει σταθερή σε όλο το πρόγραμμα.

5.5.3 Συναρτήσεις για τον υπολογισμό της συνάρτησης f(x,y)

Παρακάτω παρουσιάζονται τρεις συναρτήσεις που έχουν να κάνουν με την φόρμουλα βελτιστοποίησης που παρουσιάζεται στο τρίτο κεφάλαιο του paper, δηλαδή με την

υλοποίηση των συναρτήσεων $f_R(x,y)$ και $f_C(x,y)$ μέσω των οποίων υλοποιείται η συνάρτηση f όπως θα δούμε παρακάτω.
Ανάλυση του τι είναι η συνάρτηση f(x,y) θα γίνει στην ανάλυση του κυρίως προγράμματος και αλγόριθμου παρακάτω.

double f_R(double **R, int row_size, int col_size, double **x, double **y)
Η συνάρτηση αυτή υλοποιεί την συνάρτηση $\mathbf{f_R(x,y) = max(Ry) - x^TRy}$, που αποτελεί ένα από τα δύο συστατικά της συνάρτησης f(x,y)
[$f(x,y) = \max\{f_R(x,y), f_C(x,y)\}$].

Παίρνει ως πρώτο όρισμα τον πίνακα κερδών του παίκτη γραμμής R μεγέθους row_size x col_size, ως δεύτερο όρισμα τον αριθμό των γραμμών (row_size) του πίνακα κερδών R και ως τρίτο όρισμα τον αριθμό των στηλών (col_size) του πίνακα κερδών R.
Ως τέταρτο όρισμα παίρνει το διάνυσμα πιθανοτήτων x [κατανομή πιθανοτήτων πάνω στις στρατηγικές του παίκτη γραμμής] και ως πέμπτο όρισμα το διάνυσμα πιθανοτήτων y [κατανομή πιθανοτήτων πάνω στις στρατηγικές του παίκτη στήλης].

Στην συνάρτηση αρχικά υπολογίζουμε όλους τους απαραίτητους πίνακες που θα χρειαστούν για τον υπολογισμό της τιμής της $f_R(x,y)$.
Πρώτα υπολογίζεται ο ανάστροφος πίνακας του x [x^T], μέσω της συνάρτηση tran_matrix.
Στη συνέχεια υπολογίζεται το διάνυσμα που προκύπτει από τον πολλαπλασιασμό του πίνακα κερδών R με το διάνυσμα πιθανοτήτων y [Ry] μέσω της συνάρτησης m_v_mul.
Στη συνέχεια υπολογίζεται το στοιχείο x^TRy μέσω της συνάρτησης v_v_mul_2.
Αφού βρούμε και το μέγιστο στοιχείο max του πίνακα Ry μέσω της συνάρτησης find_max, τότε για να βρούμε την τελική τιμή του $f_R(x,y)$, απλά εκτελούμε την πράξη: max - x^TRy

Η συνάρτηση αυτή επιστρέφει αυτήν την τελική τιμή της $f_R(x,y)$.

double f_C(double **C, int row_size, int col_size, double **x, double **y)
Η συνάρτηση αυτή υλοποιεί την συνάρτηση $\mathbf{f_C(x,y) = max(C^Tx) - x^TCy}$, που αποτελεί το δεύτερο από τα δύο συστατικά της συνάρτησης f(x,y)
[$f(x,y) = \max\{f_R(x,y), f_C(x,y)\}$].

Παίρνει ως πρώτο όρισμα τον πίνακα κερδών του παίκτη στήλης C μεγέθους row_size x col_size, ως δεύτερο όρισμα τον αριθμό των γραμμών (row_size) του πίνακα κερδών C και ως τρίτο όρισμα τον αριθμό των στηλών (col_size) του πίνακα κερδών C.
Ως τέταρτο όρισμα παίρνει το διάνυσμα πιθανοτήτων x [κατανομή πιθανοτήτων πάνω στις στρατηγικές του παίκτη γραμμής] και ως πέμπτο όρισμα το διάνυσμα πιθανοτήτων y [κατανομή πιθανοτήτων πάνω στις στρατηγικές του παίκτη στήλης].

Στην συνάρτηση αρχικά υπολογίζουμε όλους τους απαραίτητους πίνακες που θα χρειαστούν για τον υπολογισμό της τιμής της $f_C(x,y)$.

Πρώτα υπολογίζεται ο ανάστροφος πίνακας του x [x^T], μέσω της συνάρτηση tran_matrix.
Στη συνέχεια υπολογίζεται ο ανάστροφος πίνακας του C [C^T], μέσω της συνάρτηση tran_matrix
Στη συνέχεια υπολογίζεται το διάνυσμα που προκύπτει από τον πολλαπλασιασμό του πίνακα C^T με το διάνυσμα πιθανοτήτων x [C^T x] μέσω της συνάρτησης m_v_mul.
Υπολογίζεται το διάνυσμα Cy μέσω της συνάρτησης m_v_mul και το στοιχείο x^TCy μέσω της συνάρτησης v_v_mul_2.
Αφού βρούμε και το μέγιστο στοιχείο max του πίνακα C^T x μέσω της συνάρτησης find_max, τότε για να βρούμε την τελική τιμή του $f_C(x,y)$, απλά εκτελούμε την πράξη: max - x^TCy

Η συνάρτηση αυτή επιστρέφει αυτήν την τελική τιμή της $f_C(x,y)$.

double f(double **R, double **C, int row_size, int col_size, double **x, double **y)
Η συνάρτηση αυτή υλοποιεί την συνάρτηση **f(x,y)**, μέσω των δύο συστατικών ($f_R(x,y)$, $f_C(x,y)$) που υπολογίζονται από τις εκάστοτε συναρτήσεις f_R και f_C αντίστοιχα.
[$f(x,y) = \max\{f_R(x,y), f_C(x,y)\}$].

Παίρνει ως πρώτο όρισμα τον πίνακα κερδών του παίκτη γραμμής R μεγέθους row_size x col_size, ως δεύτερο όρισμα τον πίνακα κερδών του παίκτη στήλης C μεγέθους row_size x col_size, ως τρίτο όρισμα τον αριθμό των γραμμών (row_size) του πίνακα κερδών R και C και ως τέταρτο όρισμα τον αριθμό των στηλών (col_size) του πίνακα κερδών R και C.
Ως πέμπτο όρισμα παίρνει το διάνυσμα πιθανοτήτων x [κατανομή πιθανοτήτων πάνω στις στρατηγικές του παίκτη γραμμής] και ως έκτο όρισμα το διάνυσμα πιθανοτήτων y [κατανομή πιθανοτήτων πάνω στις στρατηγικές του παίκτη στήλης].

Στην συνάρτηση αρχικά καλούμαι την συνάρτηση f_R με όρισμα τον πίνακα κερδών του παίκτη γραμμής R. Αποθηκεύουμε το αποτέλεσμα στον double αριθμό max_R.
Επίσης καλούμαι την συνάρτηση f_C με όρισμα τον πίνακα κερδών του παίκτη στήλης C. Αποθηκεύουμε το αποτέλεσμα στον double αριθμό max_C.
Εν συνεχεία απλώς κάνουμε έναν έλεγχο ποιο από τους δύο παραπάνω αριθμούς (max_R ή max_C) είναι ο μεγαλύτερος.

Η συνάρτηση f επιστρέφει τον μεγαλύτερο από αυτούς τους δύο, μιας και η f(x,y) ορίζεται ως εξής:

$$\mathbf{f(x,y) = \max\{f_R(x,y), f_C(x,y)\}}$$

5.6 Επίλυση LP μέσω της CPLEX

[*Η παρακάτω ανάλυση γίνεται βάσει του manual που δίνει η CPLEX: gscplex.pdf*]

Ο αλγόριθμός μας βασίζεται κατά βάση στην επίλυση γραμμικών προβλημάτων [LP]. Στο πρόγραμμα μας χρησιμοποιούμε το εργαλείο CPLEX τόσο για να ορίσουμε όσο και για να λύσουμε τα γραμμικά προβλήματα που παρουσιάζονται στον αλγόριθμο.

Όπως είπαμε και παραπάνω η CPLEX δουλεύει μέσω της γλώσσας C, απλώς προσθέτοντας την βιβλιοθήκη <ilcplex/cplex.h> (ILOG CPLEX Callable Library), η οποία περιέχει διάφορες συναρτήσεις της CPLEX [routines], τις οποίες μπορούμε να χρησιμοποιήσουμε. Μεταξύ αυτών είναι η δημιουργία και η επίλυση γραμμικών προβλημάτων.

Για αυτόν τον λόγο παρακάτω θα εξηγήσουμε πως λειτουργεί η όλη διαδικασία την οποία θα χρησιμοποιήσουμε για κάθε LP του αλγόριθμου.

5.6.1 Compile και Εκτέλεση

Έχουμε δημιουργήσει ένα makefile μέσω του οποίου γίνεται η σύνδεση του προγράμματος και της βιβλιοθήκης <ilcplex/cplex.h> η οποία υπάρχει στον server του συστήματος πάνω στο οποίο τρέχει η CPLEX [χρησιμοποιούμε τον server zenon της σχολής].
Έτσι εκτελώντας ένα απλό **make all** γίνεται το compile του κώδικα.
[Αν έχουμε ξανακάνει compile πρέπει να κάνουμε **make clean** για να σβηστούν τυχόν παλαιότερα αρχεία που είχαν δημιουργηθεί]

Έπειτα απλά τρέχουμε το εκτελέσιμο που προκύπτει.

Σημείωση: Το αρχείο makefile υπήρχε έτοιμο στον φάκελο CPLEX του server. Το μόνο που κάναμε ήταν να αλλάξουμε την διεύθυνση του αρχείου που θέλουμε να κάνουμε compile και να το προσαρμόσουμε στο πρόβλημά μας.

5.6.2 Δημιουργία και Επίλυση LP

5.6.2.1 Σύνδεση με το περιβάλλον της CPLEX (μεταβλητή env)

Για να χρησιμοποιήσουμε ρουτίνες της CPLEX βιβλιοθήκης, αρχικά πρέπει να καλέσουμε την συνάρτηση CPXopenCPLEX [αυτή πρέπει να είναι η πρώτη συνάρτηση που καλούμε] η οποία αρχικά ελέγχει την ύπαρξη κάποιου έγκυρου ILOG CPLEX license και στη συνέχεια επιστρέφει ένα δείκτη (με όνομα **env**) στο περιβάλλον CPLEX.
Ο δείκτης αυτός πρέπει να περνάει σε κάθε συνάρτηση της βιβλιοθήκης της CPLEX (Callable Library) που θα χρησιμοποιούμε στη συνέχεια.
Επίσης χρησιμοποιείται ένα όρισμα (status) το οποίο είναι δείκτης σε ακέραιο, έτσι ώστε όταν έχουμε κάποιο error να τίθεται ένας συγκεκριμένος αριθμός στο status.

5.6.2.2 Ενεργοποίηση εκτύπωσης αποτελεσμάτων

Στη συνέχεια αν θέλουμε τα αποτελέσματα της επίλυσης γραμμικών προβλημάτων να εμφανίζονται στην οθόνη απλά καλούμε την συνάρτηση CPXsetintparam δίνοντας ως όρισμα το CPX_PARAM_SCRIND.
Αν δεν τεθεί η συγκεκριμένη επιλογή τότε δεν μπορούμε να δούμε ούτε στην οθόνη ούτε σε αρχείο τα αποτελέσματα της επίλυσης του LP.

5.6.2.3 Δημιουργία νέου Γραμμικού Προβλήματος (μεταβλητή lp)

Εν συνεχεία καλώντας την συνάρτηση CPXcreateprob δημιουργούμε ένα κενό LP πρόβλημα το οποίο by default είναι πρόβλημα ελαχιστοποίησης, χωρίς καθόλου αντικειμενική συνάρτηση, περιορισμούς και μεταβλητές. Αυτό το πρόβλημα πρέπει

να το διαμορφώσουμε στη συνέχεια και να το προσαρμόσουμε στα δεδομένα μας, μέσω συναρτήσεων της CPLEX που θα δούμε παρακάτω.
Η επιστρεφόμενη τιμή (**lp**) της συνάρτησης αυτής, είναι ένας δείκτης που πρέπει να περνάει ως όρισμα σε οποιαδήποτε άλλη συνάρτηση ώστε να σηματοδοτείται ότι μιλάμε για το συγκεκριμένο πρόβλημα.
Αν δεν επιτύχει η δημιουργία του LP αυτού, τότε επιστρέφεται η τιμή NULL.

5.6.2.4 Καθορισμός Παραμέτρων του LP μας (συνάρτηση populatebyrow)

Στη συνέχεια καλούμε μια συνάρτηση που έχουμε δημιουργήσει εμείς (με όνομα που αρχίζει με populatebyrow), η οποία δημιουργεί το LP που θέλουμε.

Στην συνάρτηση αυτήν, αρχικά καθορίζουμε **το είδος του προβλήματος** (maximixzation - minimization) μέσω της συνάρτησης CPXchgobjsen.
Έτσι αν το όρισμα είναι CPX_MIN τότε το γραμμικό πρόβλημα θα είναι πρόβλημα ελαχιστοποίησης της αντικειμενικής συνάρτησης (minimization), ενώ αν το όρισμα είναι CPX_ MAX τότε το γραμμικό πρόβλημα θα είναι πρόβλημα μεγιστοποίησης της αντικειμενικής συνάρτησης (maximization).

Στη συνέχεια καθορίζουμε την **αντικειμενική συνάρτηση** του LP.
Οι αντικειμενικές συναρτήσεις που θα χρησιμοποιήσουμε θα είναι της μορφής:

$$\mathbf{a_1*x_1 + a_2*x_2 + a_3*x_3 + \dots}$$

όπου $\mathbf{x_i}$ είναι οι μεταβλητές της αντικειμενικής συνάρτησης και $\mathbf{a_i}$ είναι ο συντελεστής κάθε μιας μεταβλητής αντίστοιχα.
Για να το καθορίσουμε αυτό αρχικά χρησιμοποιούμε κάποιους πίνακες που περιέχουν double στοιχεία. Αυτοί οι πίνακες είναι οι **obj, lb, ub** μεγέθους ίσο με το πλήθος των μεταβλητών της αντικειμενικής συνάρτησης. Κάθε θέση των πινάκων αυτών, ξεκινώντας από την αρχή, αντιπροσωπεύει και μία μεταβλητή.
Έτσι για να καταχωρήσουμε την μεταβλητή x_i ως μεταβλητή της αντικειμενικής συνάρτησης στη θέση i με συντελεστή a_i αρκεί να κάνουμε το εξής:

obj[i] = ai
lb[i] = (εδώ βάζουμε το άνω όριο της τιμής που μπορεί να πάρει η μεταβλητή x_i)
ub[i] = (εδώ βάζουμε το κάτω όριο της τιμής που μπορεί να πάρει η μεταβλητή x_i)

Αφού κάνουμε αυτή την διαδικασία για κάθε μεταβλητή της αντικειμενικής συνάρτησης καλούμε την συνάρτηση CPXnewcols η οποία παίρνοντας ως ορίσματα το πλήθος των μεταβλητών, τους πίνακες obj, lb, ub καθορίζει την αντικειμενική συνάρτηση του προβλήματος που καθορίζεται μέσω των μεταβλητών env, lp όπως είδαμε παραπάνω.

Στη συνέχεια καθορίζουμε το σύνολο των **περιορισμών** (αν υπάρχουν περιορισμοί, πέραν αυτούς που βάλαμε κατά τη δημιουργία της αντικειμενικής συνάρτησης).
Οι περιορισμοί καθορίζονται μέσω τριών πινάκων.

ΠΡΟΣΟΧΗ: Πρέπει να πούμε ότι οι μεταβλητές που εμφανίζονται στους περιορισμούς πρέπει να αντιστοιχούν στις μεταβλητές που εμφανίστηκαν στην αντικειμενική συνάρτηση, με την ίδια ακριβώς σειρά.

Δηλαδή, συνεχίζοντας το παραπάνω παράδειγμα οι περιορισμοί πρέπει να είναι της μορφής:

$$\mathbf{b_1*x_1 + b_2*x_2 + b_3*x_3 + \ldots \geq (\acute{\eta} \leq) B}$$

όπου $\mathbf{x_i}$ είναι οι ίδιες μεταβλητές που εμφανίστηκαν στην αντικειμενική συνάρτηση, $\mathbf{b_i}$ είναι ο συντελεστής κάθε μιας μεταβλητής στους περιορισμούς αντίστοιχα και το **B** είναι ένας σταθερός αριθμός.
Μπορούμε να έχουμε όσους τέτοιους περιορισμούς επιθυμούμε, αλλά πάντα στη συγκεκριμένη μορφή.
Παραδείγματος χάρη να έχουμε δύο τέτοιους περιορισμούς, που έχουν τρεις μεταβλητές ο κάθε ένας:

$$b_1*x_1 + b_2*x_2 + b_3*x_3 \leq B$$
$$c_1*x_1 + c_2*x_2 + c_3*x_3 \leq C$$

Ο πίνακας **rmatind** είναι ένας πίνακας μεγέθους ίσο με το σύνολο όλων των μεταβλητών που εμφανίζονται στους περιορισμούς (δηλαδή ίσο με το γινόμενο μεταξύ του πλήθους των περιορισμών και το πλήθος των μεταβλητών που ένας περιορισμός έχει)
Έτσι όταν έχουμε rmatind[i] αναφερόμαστε στην i-στη μεταβλητή μετρώντας τις μεταβλητές όλων των περιορισμών από την αρχή.
Για να καθορίσουμε ότι μία μεταβλητή είναι η j-στη κατά σειρά σε ένα συγκεκριμένο περιορισμό (και η i-στη συνολικά στους περιορισμούς) θέτουμε:

rmatind[i] = j

Επίσης ο πίνακας **rmatval** είναι ίδιου μεγέθους με τον rmatind, μόνο που σε αυτόν καθορίζουμε την τιμή του συντελεστή που έχει η i-στη (συνολικά στους περιορισμούς) μεταβλητή:

rmatval[i] = (value)

Ο **rmatbeg** είναι ένας πίνακας μεγέθους όσο και το πλήθος των περιορισμών.
Κάθε θέση του εκφράζει τον αντιστοιχο κατά σειρά περιορισμό. Δηλαδή αν θέλουμε να αναφερθούμε στον k-στο κατά σειρά περιορισμό τότε γράφουμε:

rmatbeg[k]

Έτσι ο rmatbeg[k] είναι ουσιαστικά ένας "δείκτης" στον k κατά σειρά περιορισμό που θέτουμε.
Η τιμή που παίρνει η rmatbeg[k] είναι ίση με τον αριθμό της θέσης που έχει η πρώτη μεταβλητή του περιορισμού k και φαίνεται στον πίνακα rmatind.

Με άλλα λόγια για τον i-στο κατά σειρά περιορισμό [θα λέμε ότι είναι ο περιορισμός της γραμμής i] στο πίνακα rmatind αποθηκεύουμε τον αριθμό της στήλης (θα αναφερόμαστε στις μεταβλητές ως στήλες. Η πρώτη μεταβλητή κατέχει την πρώτη στήλη) που περιέχει μη μηδενική μεταβλητή.

Δηλαδή στην γραμμή i οι στήλες αποθηκεύονται στα:
rmatind[rmatbeg[i]], rmatind[rmatbeg[i]+1],..., rmatind[rmatbeg[i+1]-1].

Και οι τιμές των στηλών στα:
rmatval[rmatbeg[i]], rmatval[rmatbeg[i]+1],..., rmatval[rmatbeg[i+1]-1].

Επίσης μπορούμε να καθορίσουμε την φορά της ανισότητας μέσω του πίνακα **sense[k]** για τον k κατά σειρά περιορισμό θέτοντας το ίσο είτε με L (≤), είτε ίσο με H (≥), είτε ίσο με E (=).

Επίσης μπορούμε να καθορίσουμε την τιμή της σταθερής τιμής μέσω του πίνακα **rhs[k]** για τον k κατά σειρά περιορισμό θέτοντας τον απλά ίσο με την τιμή που θέλουμε.

Στο παράδειγμα:

$$b_1*x_1 + b_2*x_2 + b_3*x_3 \leq B$$
$$c_1*x_1 + c_2*x_2 + c_3*x_3 \leq C$$

οι πίνακες θα έχουν τις εξής τιμές:

[Για τον πρώτο περιορισμό]

rmatbeg[0] = 0		
sense[0] = 'L'		
rhs[0] = B		
rmatind[0] = 0	rmatind[1] = 1	rmatind[2] = 2
rmatval [0] = b_1	rmatval [1] = b_2	rmatval [2] = b_3

[Για τον δεύτερο περιορισμό]

rmatbeg[1] = 3		
sense[1] = 'L'		
rhs[1] = C		
rmatind[3] = 0	rmatind[4] = 1	rmatind[5] = 2
rmatval [3] = c_1	rmatval [4] = c_2	rmatval [5] = c_3

Αφού καθορίσουμε τους περιορισμούς μέσω των πινάκων rmatbeg, rmatind, rmatval, sense και rhs καλούμε την συνάρτηση CPXnewcols η οποία παίρνοντας ως ορίσματα αυτούς τους πίνακες καθορίζει τους περιορισμούς του προβλήματος που καθορίζεται μέσω των μεταβλητών env, lp όπως είδαμε παραπάνω.

5.6.2.5 Η Επίλυση του Γραμμικού Προβλήματος (optimization)

Αφού έχουμε καθορίσει τόσο την αντικειμενική συνάρτηση του LP, όσο και τους περιορισμούς, πλέον μπορούμε να κάνουμε το optimization του LP που έχουμε δημιουργήσει. Αυτό γίνεται μέσω της συνάρτησης CPXlpopt η οποία παίρνει ορίσματα τις μεταβλητές env και lp.
By default χρησιμοποιείται ο Optimizer που δουλεύει με την μέθοδο **Dual Simplex**.
Η επιστρεφόμενη τιμή αν λειτουργήσει σωστά είναι 0.

Αν όλα λειτουργήσουν σωστά και γίνει το optimization τότε η CPLEX δεσμεύει χώρο για τις primal μεταβλητές, τις dual και την τιμή της αντικειμενικής συνάρτησης (καθώς και κάποιες άλλες παραμέτρους που δεν μας ενδιαφέρουν στο πρόγραμμά μας).

Τότε καλούμε την συνάρτηση CPXsolution, μέσω της οποίας μπορούμε να έχουμε πρόσβαση στις τιμές αυτές που θέλουμε.
Η συνάρτηση αυτήν επιστρέφει την κατάσταση του προβλήματος (αν υπάρχει λύση ή είναι αδύνατο να λυθεί), την τιμή της αντικειμενικής συνάρτησης και τα διανύσματα με τις λύσεις (τις τιμές των primal και dual μεταβλητών)

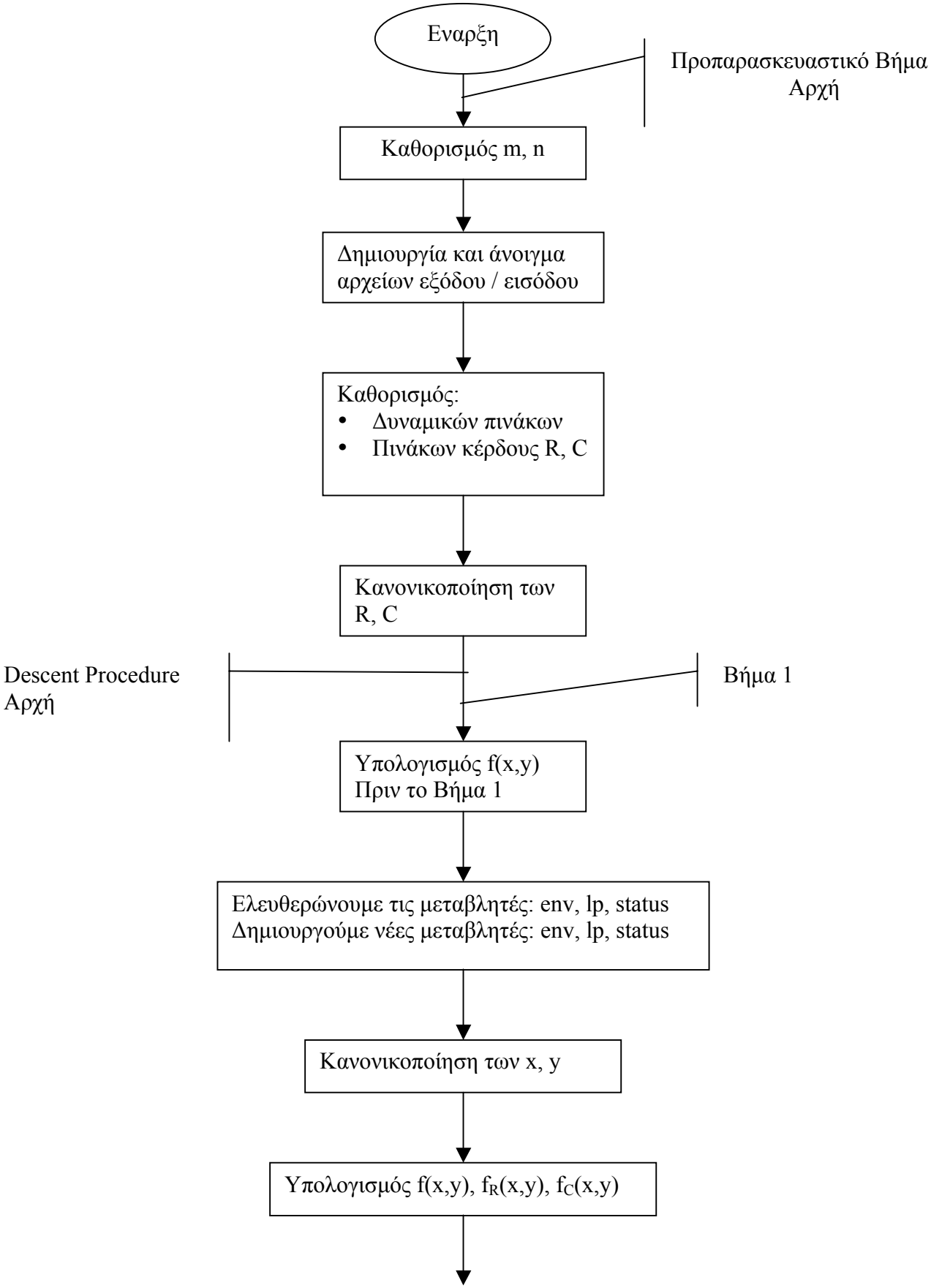
Μπορούμε να γράψουμε τα του προβλήματος σε κάποιο αρχείο μέσω της συνάρτησης CPXwriteprob, ώστε να κάνουμε κάποιο debugging αν είναι απαραίτητο.

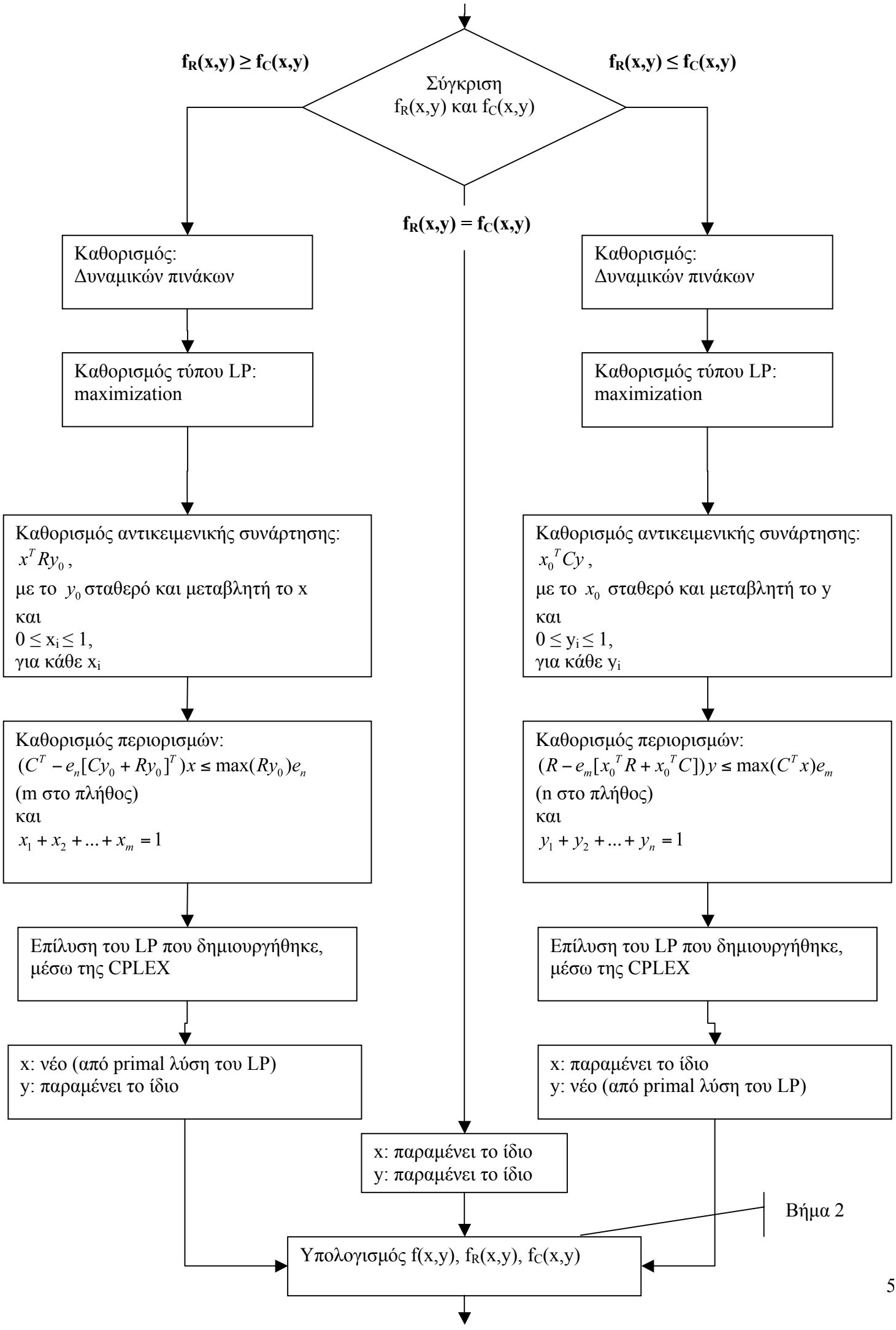
5.6.2.6 Απελευθέρωση του δεσμευμένου χώρου λύσεων και Τερματισμός

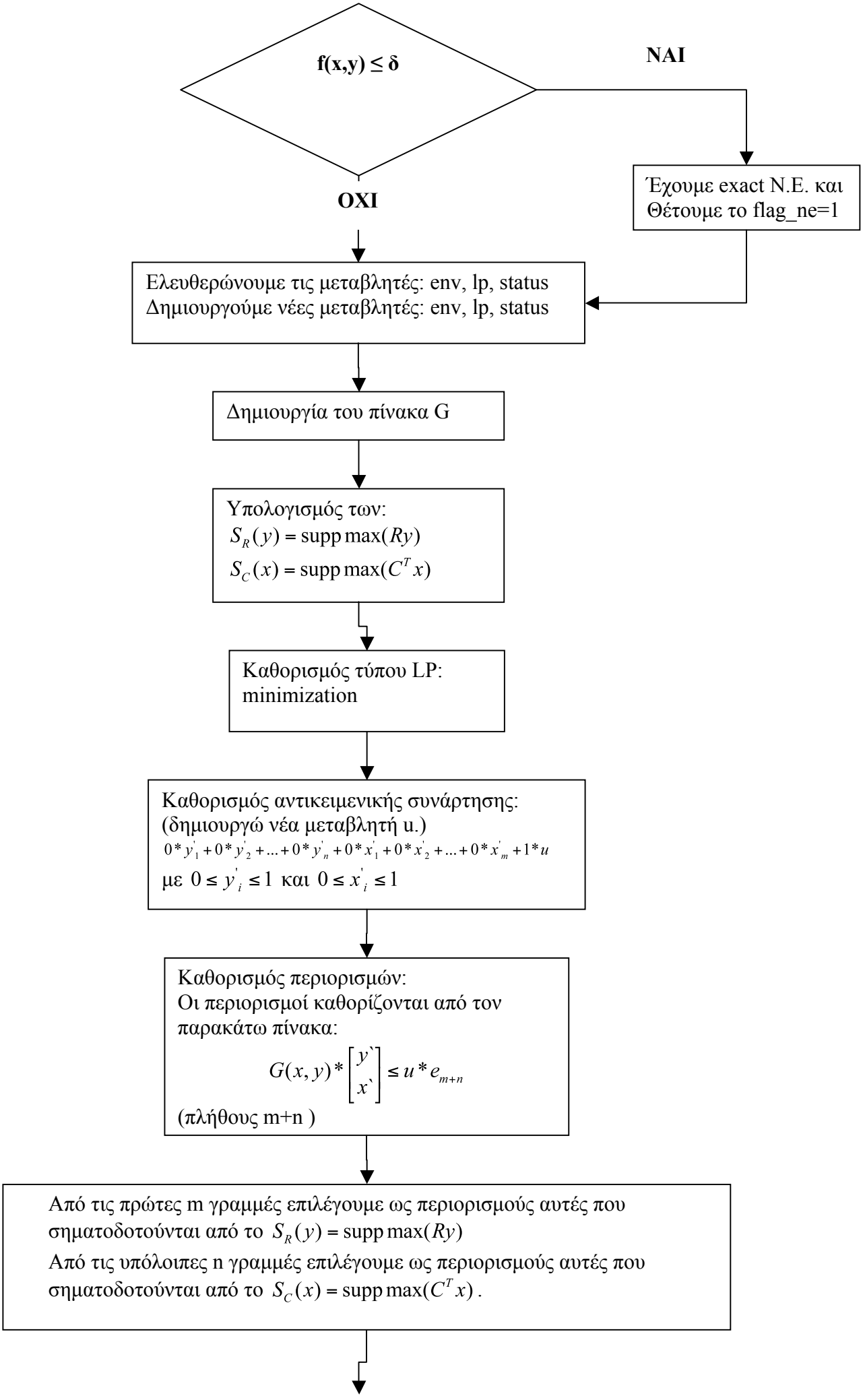
Τέλος εφόσον έχουμε τελειώσει με το LP το οποίο είχαμε να λύσουμε ΠΡΕΠΕΙ να ελευθερώσουμε τις μεταβλητές env και lp πριν πάμε να λύσουμε κάποιο άλλο LP. Την μεταβλητή lp, που αντιπροσωπεύει το πρόγραμμα, την ελευθερώνουμε μέσω της συνάρτησης CPXfreeprob και την μεταβλητή env, που αντιπροσωπεύει την σύνδεση με το περιβάλλον της CPLEX, μέσω της συνάρτησης CPXcloseCPLEX.
Έτσι πλέον δεν υφίσταται το LP που είχαμε δημιουργήσει και μπορούμε να δημιουργήσουμε κάποιο νέο.

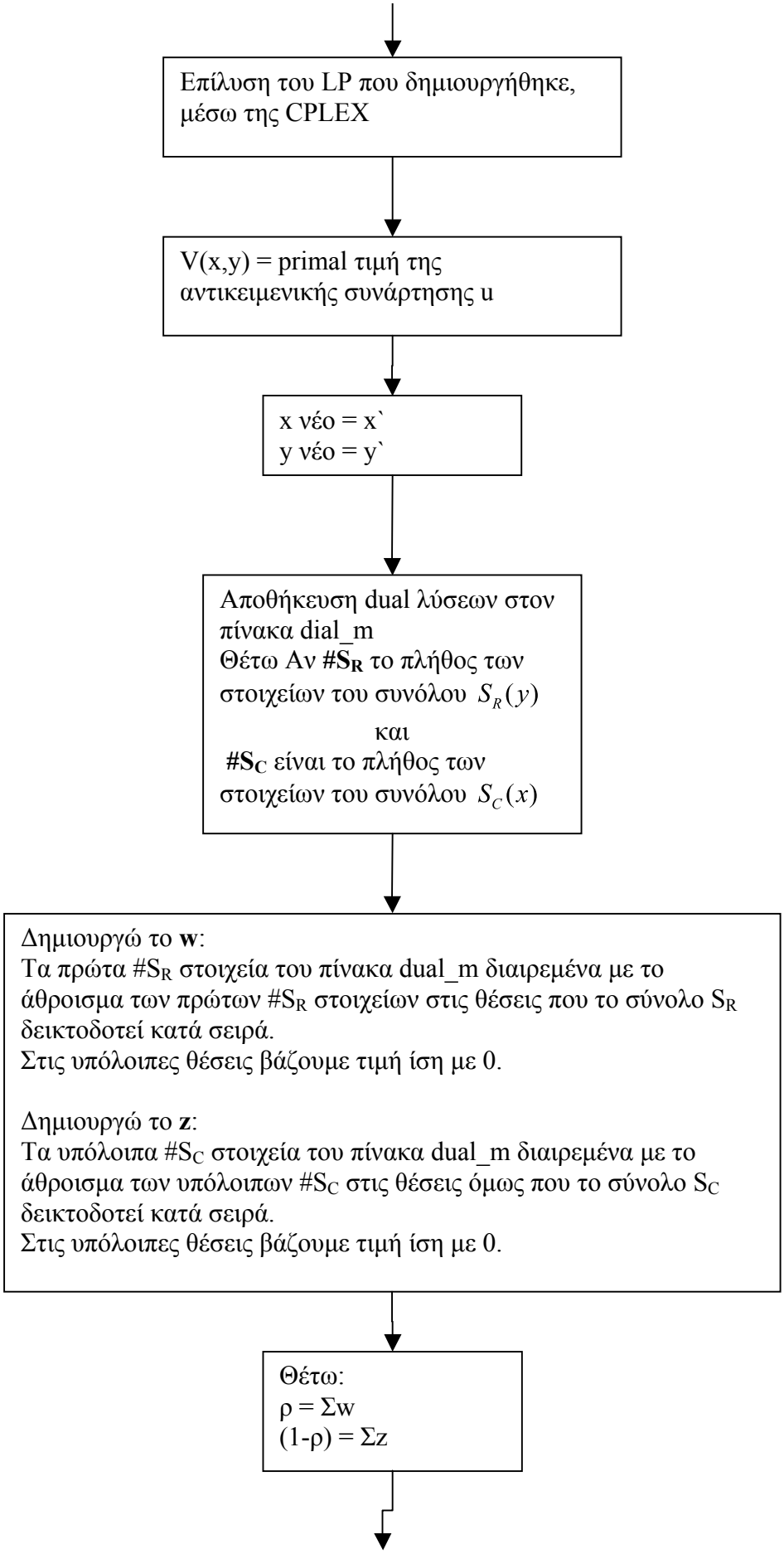
Επίσης η ετικέτα TERMINATE: υφίσταται ώστε αν συμβεί κάποιο σφάλμα στο πρόγραμμα (το ελέγχουμε μέσω της μεταβλητή status) να τερματίσουμε το πρόγραμμα κανονικά.
Έτσι μετά την ετικέτα αυτήν το αντικείμενο lp που αντιπροσωπεύει το πρόγραμμα τερματίζει (με την CPXfreeprob) και επίσης απελευθερώνονται όλοι οι δυναμικοί πίνακες που είχαν δεσμευτεί για τις λύσεις.
Τέλος όπως είδαμε μέσω της CPXcloseCPLEX ενημερώνεται η CPLEX ότι όλες οι κλήσεις στην βιβλιοθήκη (Callable Library) έχουν ολοκληρωθεί.

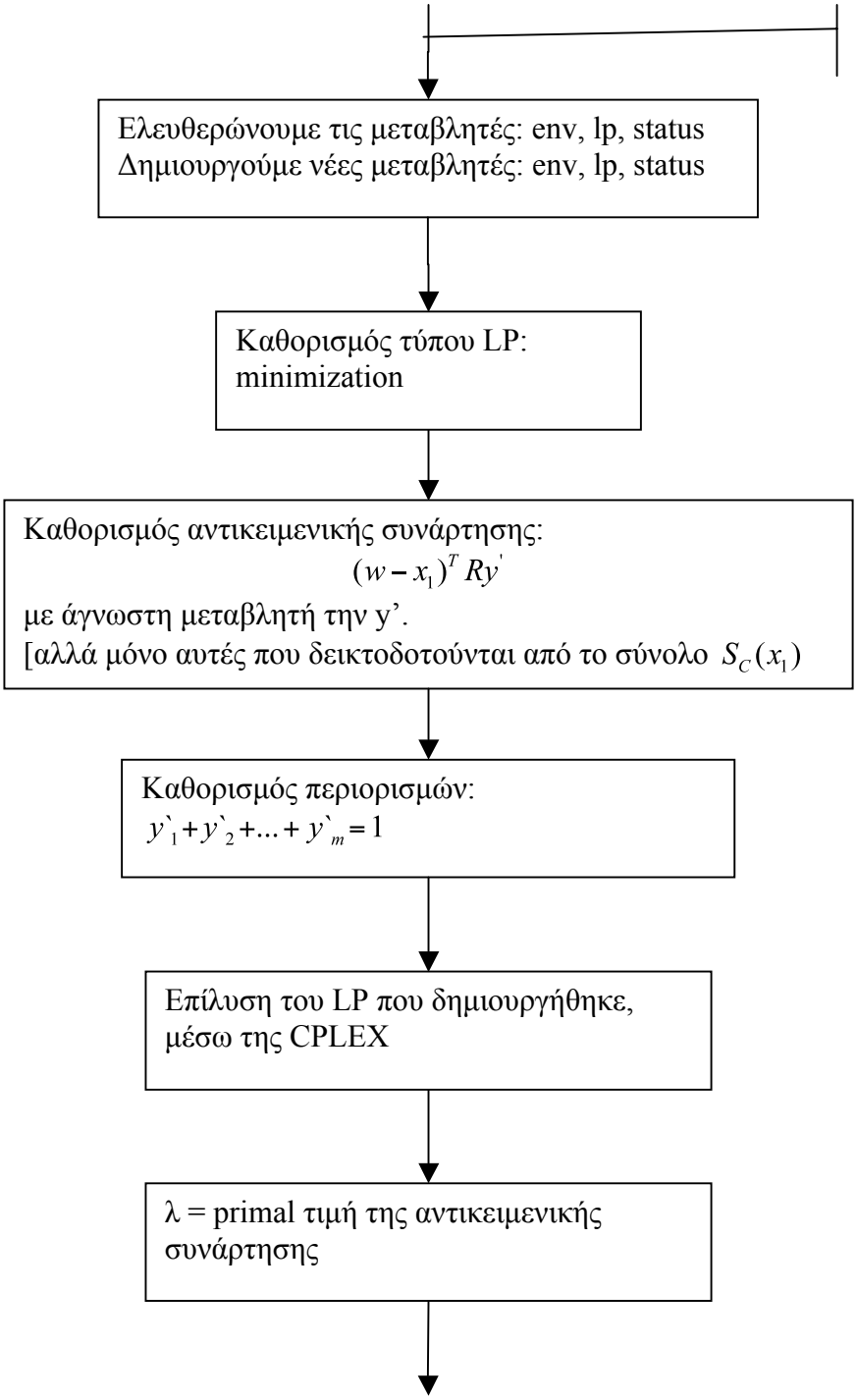
5.7 Αλγόριθμος – Descent Procedure (σχηματικά)

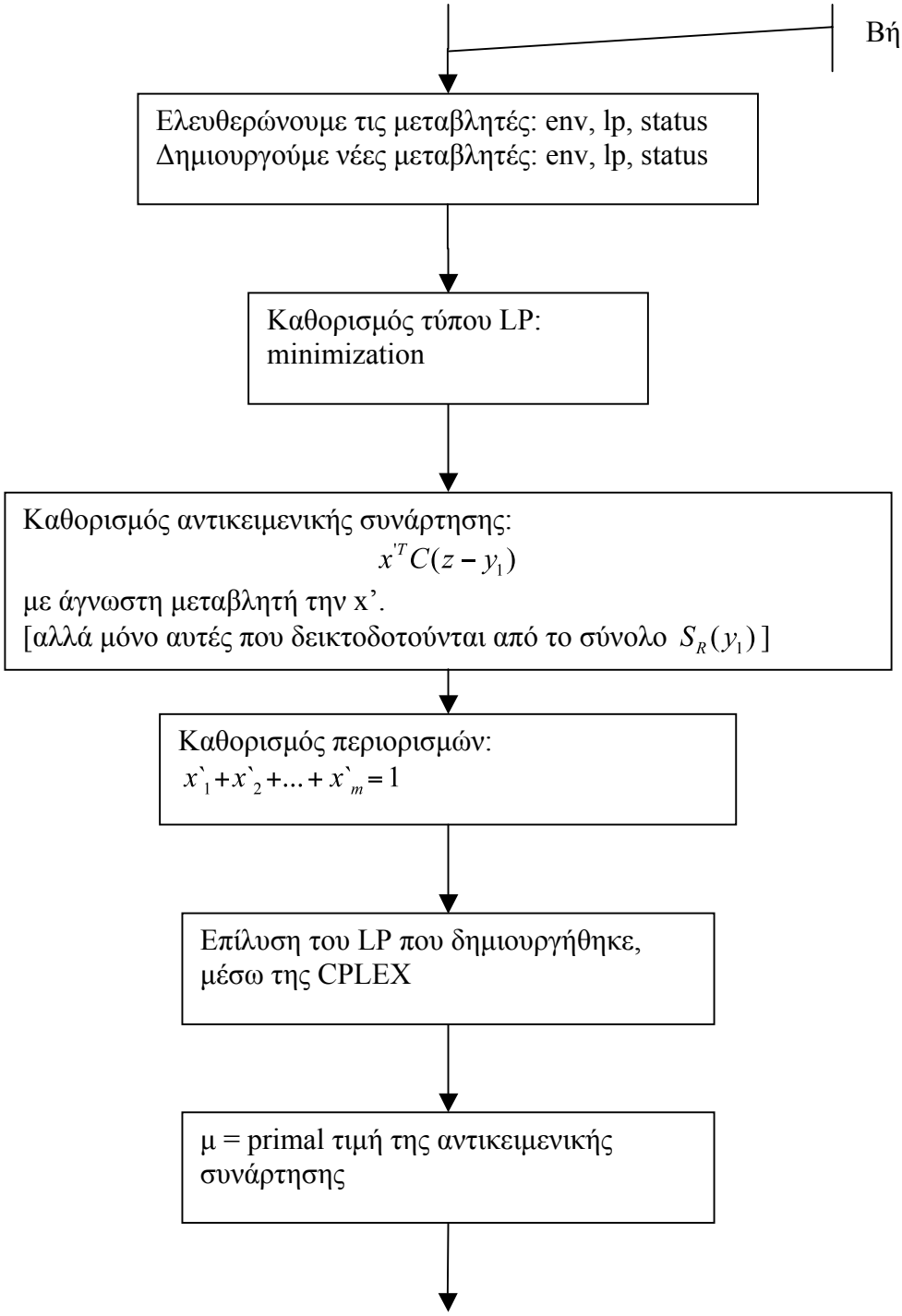

Εναρξη
Προπαρασκευαστικό Βήμα Αρχή
Καθορισμός m, n
Δημιουργία και άνοιγμα αρχείων εξόδου / εισόδου
Καθορισμός:
• Δυναμικών πινάκων
• Πινάκων κέρδους R, C
Κανονικοποίηση των R, C
Descent Procedure Αρχή
Βήμα 1
Υπολογισμός f(x,y) Πριν το Βήμα 1
Ελευθερώνουμε τις μεταβλητές: env, lp, status
Δημιουργούμε νέες μεταβλητές: env, lp, status
Κανονικοποίηση των x, y
Υπολογισμός f(x,y), f_R(x,y), f_C(x,y)


$f_R(x,y) \geq f_C(x,y)$
Σύγκριση
$f_R(x,y)$ και $f_C(x,y)$
$f_R(x,y) \leq f_C(x,y)$
$f_R(x,y) = f_C(x,y)$
Καθορισμός:
Δυναμικών πινάκων
Καθορισμός τύπου LP:
maximization
Καθορισμός αντικειμενικής συνάρτησης:
$x^T R y_0$,
με το y_0 σταθερό και μεταβλητή το x
και
$0 \leq x_i \leq 1$,
για κάθε x_i
Καθορισμός περιορισμών:
$(C^T - e_n[Cy_0 + Ry_0]^T)x \leq \max(Ry_0)e_n$
(m στο πλήθος)
και
$x_1 + x_2 + ... + x_m = 1$
Επίλυση του LP που δημιουργήθηκε,
μέσω της CPLEX
x: νέο (από primal λύση του LP)
y: παραμένει το ίδιο
Καθορισμός:
Δυναμικών πινάκων
Καθορισμός τύπου LP:
maximization
Καθορισμός αντικειμενικής συνάρτησης:
${x_0}^T Cy$,
με το x_0 σταθερό και μεταβλητή το y
και
$0 \leq y_i \leq 1$,
για κάθε y_i
Καθορισμός περιορισμών:
$(R - e_m[{x_0}^T R + {x_0}^T C])y \leq \max(C^T x)e_m$
(n στο πλήθος)
και
$y_1 + y_2 + ... + y_n = 1$
Επίλυση του LP που δημιουργήθηκε,
μέσω της CPLEX
x: παραμένει το ίδιο
y: νέο (από primal λύση του LP)
x: παραμένει το ίδιο
y: παραμένει το ίδιο
Υπολογισμός f(x,y), $f_R(x,y)$, $f_C(x,y)$
Βήμα 2


f(x,y) ≤ δ
ΝΑΙ
ΟΧΙ
Έχουμε exact N.E. και
Θέτουμε το flag_ne=1
Ελευθερώνουμε τις μεταβλητές: env, lp, status
Δημιουργούμε νέες μεταβλητές: env, lp, status
Δημιουργία του πίνακα G
Υπολογισμός των:
$S_R(y) = \text{supp}\max(Ry)$
$S_C(x) = \text{supp}\max(C^T x)$
Καθορισμός τύπου LP:
minimization
Καθορισμός αντικειμενικής συνάρτησης:
(δημιουργώ νέα μεταβλητή u.)
$0*y'_1 + 0*y'_2 + \ldots + 0*y'_n + 0*x'_1 + 0*x'_2 + \ldots + 0*x'_m + 1*u$
με $0 \le y'_i \le 1$ και $0 \le x'_i \le 1$
Καθορισμός περιορισμών:
Οι περιορισμοί καθορίζονται από τον παρακάτω πίνακα:
$G(x,y) * \begin{bmatrix} y` \\ x` \end{bmatrix} \le u * e_{m+n}$
(πλήθους m+n)
Από τις πρώτες m γραμμές επιλέγουμε ως περιορισμούς αυτές που σηματοδοτούνται από το $S_R(y) = \text{supp}\max(Ry)$
Από τις υπόλοιπες n γραμμές επιλέγουμε ως περιορισμούς αυτές που σηματοδοτούνται από το $S_C(x) = \text{supp}\max(C^T x)$.


Επίλυση του LP που δημιουργήθηκε, μέσω της CPLEX
V(x,y) = primal τιμή της αντικειμενικής συνάρτησης u
x νέο = x`
y νέο = y`
Αποθήκευση dual λύσεων στον πίνακα dial_m
Θέτω Αν #S_R το πλήθος των στοιχείων του συνόλου S_R(y)
και
#S_C είναι το πλήθος των στοιχείων του συνόλου S_C(x)
Δημιουργώ το w:
Τα πρώτα #S_R στοιχεία του πίνακα dual_m διαιρεμένα με το άθροισμα των πρώτων #S_R στοιχείων στις θέσεις που το σύνολο S_R δεικτοδοτεί κατά σειρά.
Στις υπόλοιπες θέσεις βάζουμε τιμή ίση με 0.
Δημιουργώ το z:
Τα υπόλοιπα #S_C στοιχεία του πίνακα dual_m διαιρεμένα με το άθροισμα των υπόλοιπων #S_C στις θέσεις όμως που το σύνολο S_C δεικτοδοτεί κατά σειρά.
Στις υπόλοιπες θέσεις βάζουμε τιμή ίση με 0.
Θέτω:
ρ = Σw
(1-ρ) = Σz


Βήμα 2b
Ελευθερώνουμε τις μεταβλητές: env, lp, status
Δημιουργούμε νέες μεταβλητές: env, lp, status
Καθορισμός τύπου LP:
minimization
Καθορισμός αντικειμενικής συνάρτησης:
$(w - x_1)^T R y'$
με άγνωστη μεταβλητή την y'.
[αλλά μόνο αυτές που δεικτοδοτούνται από το σύνολο $S_C(x_1)$
Καθορισμός περιορισμών:
$y`_1 + y`_2 + ... + y`_m = 1$
Επίλυση του LP που δημιουργήθηκε,
μέσω της CPLEX
λ = primal τιμή της αντικειμενικής
συνάρτησης


Βήμα 2c
Ελευθερώνουμε τις μεταβλητές: env, lp, status
Δημιουργούμε νέες μεταβλητές: env, lp, status
Καθορισμός τύπου LP:
minimization
Καθορισμός αντικειμενικής συνάρτησης:
$x'^{T} C(z - y_1)$
με άγνωστη μεταβλητή την x'.
[αλλά μόνο αυτές που δεικτοδοτούνται από το σύνολο $S_R(y_1)$]
Καθορισμός περιορισμών:
$x`_1 + x`_2 + ... + x`_m = 1$
Επίλυση του LP που δημιουργήθηκε,
μέσω της CPLEX
μ = primal τιμή της αντικειμενικής
συνάρτησης

Βήμα 4

Κανονικοποίηση διανυσμάτων πιθανοτήτων:
x, y

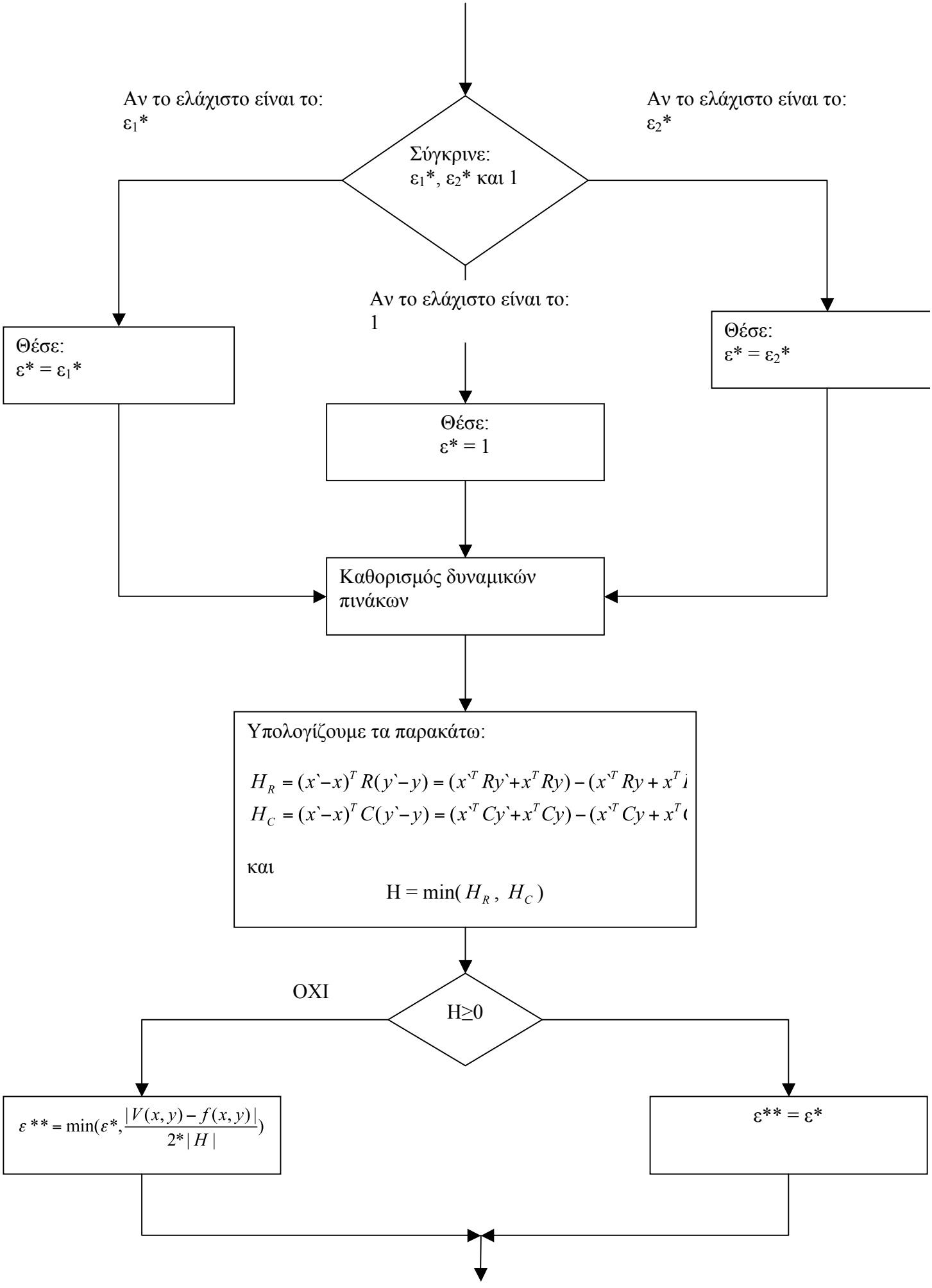
Καθορισμός δυναμικών πινάκων

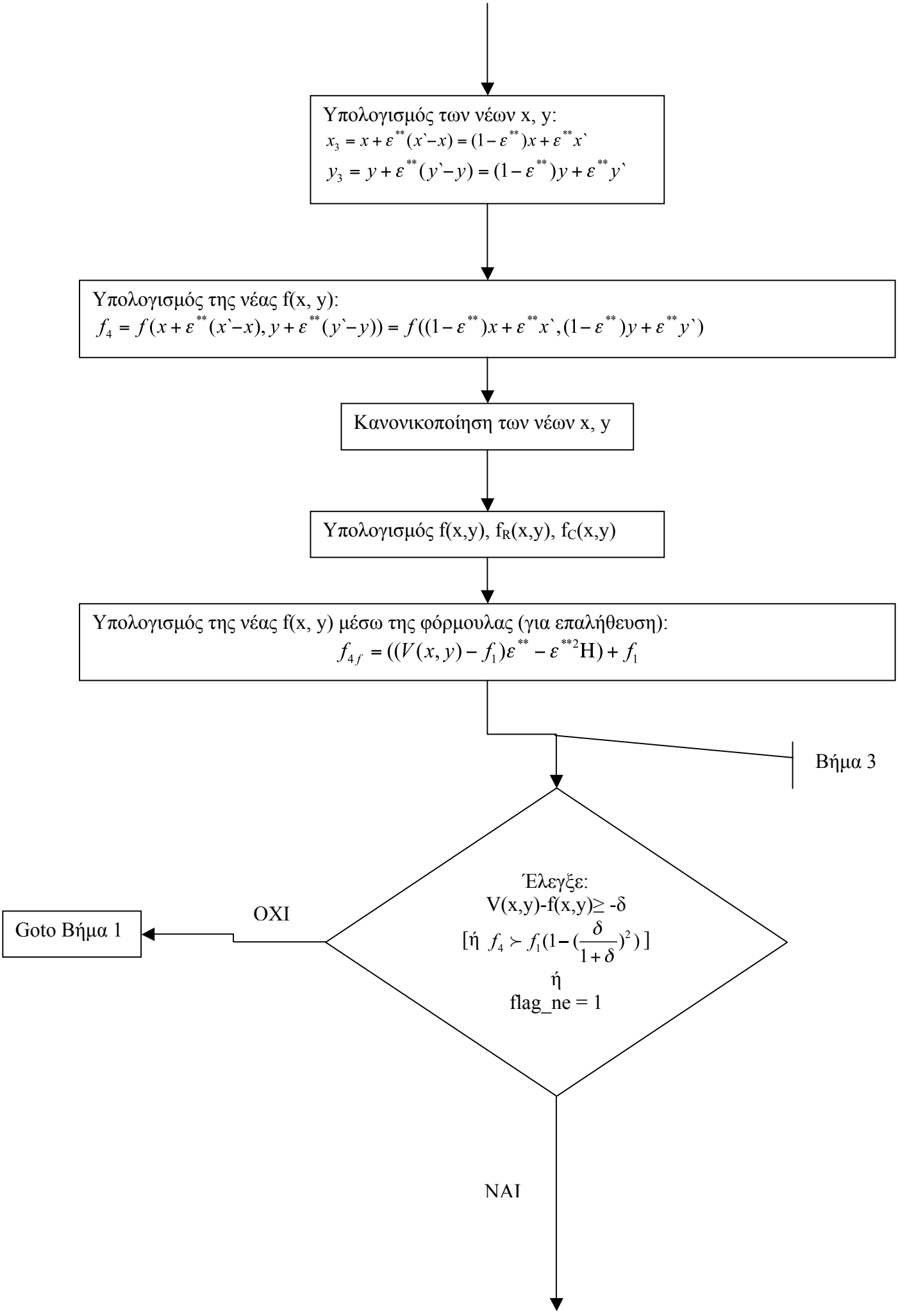
Υπολογισμός των συνόλων:
$S_R(y) = \text{supp}\max(Ry)$ και $S_C(x) = \text{supp}\max(C^T x)$
καθώς και των συμπληρωματικών αυτών:
$\overline{S}_R(y)$ και $\overline{S}_C(x)$

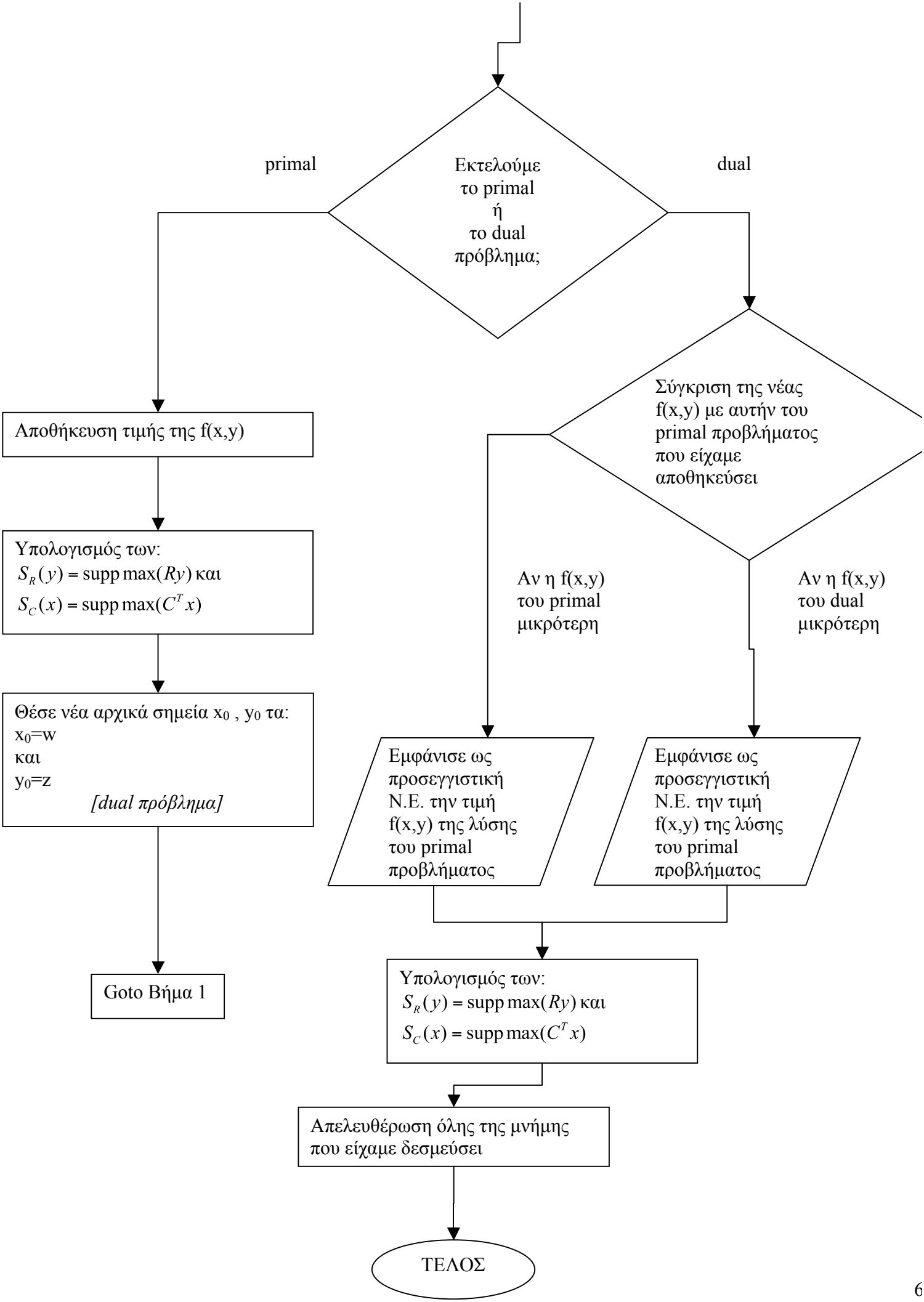
Υπολογισμός των:
$$\max(Ry)$$
καθώς επίσης και το
$$\max(Ry`),$$
για το δεύτερο, ελέγχοντας μόνο τις θέσεις που δεικτοδοτούνται από το σύνολο $S_R(y)$

Υπολογισμός του ε_1* :
$$\varepsilon_1 * = \frac{\max(Ry) - (Ry)_i}{\max(Ry) - (Ry)_i + (Ry`)_i - \max_{S_R(y)}(Ry`)}$$
για κάθε $i \in \overline{S}_R(y)$

Υπολογισμός του ε_2* :
$$\varepsilon_2 * = \frac{\max(C^T x) - (C^T x)_j}{\max(C^T x) - (C^T x)_j + (C^T x`)_j - \max_{S_C(x)}(C^T x`)}$$
για κάθε $j \in \overline{S}_C(x)$


Αν το ελάχιστο είναι το:
ε1*
Αν το ελάχιστο είναι το:
ε2*
Σύγκρινε:
ε1*, ε2* και 1
Αν το ελάχιστο είναι το:
1
Θέσε:
ε* = ε1*
Θέσε:
ε* = 1
Θέσε:
ε* = ε2*
Καθορισμός δυναμικών πινάκων
Υπολογίζουμε τα παρακάτω:
και
H = min(H_R , H_C)
ΟΧΙ
H≥0
ε** = ε*


Υπολογισμός των νέων x, y:
$x_3 = x + \varepsilon^{**}(x`-x) = (1-\varepsilon^{**})x + \varepsilon^{**}x`$
$y_3 = y + \varepsilon^{**}(y`-y) = (1-\varepsilon^{**})y + \varepsilon^{**}y`$
Υπολογισμός της νέας f(x, y):
$f_4 = f(x + \varepsilon^{**}(x`-x), y + \varepsilon^{**}(y`-y)) = f((1-\varepsilon^{**})x + \varepsilon^{**}x`, (1-\varepsilon^{**})y + \varepsilon^{**}y`)$
Κανονικοποίηση των νέων x, y
Υπολογισμός f(x,y), f_R(x,y), f_C(x,y)
Υπολογισμός της νέας f(x, y) μέσω της φόρμουλας (για επαλήθευση):
$f_{4f} = ((V(x,y) - f_1)\varepsilon^{**} - \varepsilon^{**2}\mathrm{H}) + f_1$
Βήμα 3
Έλεγξε:
V(x,y)-f(x,y)≥ -δ
[ή $f_4 \succ f_1(1-(\frac{\delta}{1+\delta})^2)$]
ή
flag_ne = 1
ΟΧΙ
Goto Βήμα 1
ΝΑΙ


Εκτελούμε το primal ή το dual πρόβλημα;
primal
dual
Αποθήκευση τιμής της f(x,y)
Υπολογισμός των:
$S_R(y) = \text{supp}\max(Ry)$ και
$S_C(x) = \text{supp}\max(C^T x)$
Θέσε νέα αρχικά σημεία x_0 , y_0 τα:
x_0=w
και
y_0=z
[dual πρόβλημα]
Goto Βήμα 1
Σύγκριση της νέας f(x,y) με αυτήν του primal προβλήματος που είχαμε αποθηκεύσει
Αν η f(x,y) του primal μικρότερη
Αν η f(x,y) του dual μικρότερη
Εμφάνισε ως προσεγγιστική N.E. την τιμή f(x,y) της λύσης του primal προβλήματος
Εμφάνισε ως προσεγγιστική N.E. την τιμή f(x,y) της λύσης του primal προβλήματος
Υπολογισμός των:
$S_R(y) = \text{supp}\max(Ry)$ και
$S_C(x) = \text{supp}\max(C^T x)$
Απελευθέρωση όλης της μνήμης που είχαμε δεσμεύσει
ΤΕΛΟΣ

5.8 Αλγόριθμος – Descent Procedure (αναλυτικά η μεθοδολογία)

Παρακάτω θα αναλυθεί λεπτομερώς ο αλγόριθμος της Descent Διαδικασίας, όπως αυτή διαφοροποιήθηκε από αυτή του paper [1] και διαμορφώθηκε στην πράξη στο πρόγραμμα [***bcs_v3.c***].

5.8.1 Αρχικές παρατηρήσεις

Ο αλγόριθμος που παρουσιάστηκε στο paper αυτό [1], χρησιμοποιεί μια εντελώς διαφορετική αλγοριθμική ιδέα από αυτές που είχαν παρουσιαστεί μέχρι τότε για την λύση του αντίστοιχου προβλήματος. Δεν στηρίζεται πάνω στην εύρεση κάποιου small support πάνω στις στρατηγικές των παικτών, όπως μέχρι τώρα παρουσιάζονταν στις λύσεις διαφόρων άλλων αλγορίθμων που έψαχναν για κατά προσέγγιση Ισορροπία Nash.
Αυτές οι προσεγγίσεις ενώ παρουσίαζαν καλή συμπεριφορά, ήταν αρκετά δύσκολο να υλοποιηθούν πρακτικά. Σε αντίθεση η νέα αυτή προσέγγιση του αλγορίθμου, όπως θα δούμε και παρακάτω, το κάνει άμεσα εφαρμόσιμο, πρακτικό και εύκολα υλοποιήσιμο σε σχέση με τις άλλες τεχνικές.
Και έτσι είναι πλέον πρακτικά δυνατόν η παρατήρηση της συμπεριφορά του αλγορίθμου και η εξαγωγή σημαντικών συμπερασμάτων από αυτήν.

Αυτό ακριβώς υλοποιήθηκε σε αυτήν την διπλωματική εργασία.

<u>Σημείωση:</u> Την ίδια χρονιά δημοσίευσης της εργασίας [1], δημοσιεύτηκε και η εργασία [26] που χρησιμοποιεί επίσης τεχνικές Γραμμικού Προγραμματισμού για την εύρεση προσεγγιστικής Ισορροπίας Nash, αλλά με ένα διαφορετικό τρόπο.
Το επιχείρημά τους ότι ο αλγόριθμος που χρησιμοποιούμε στην παρούσα εργασία χρειάζεται την λύση πολυωνυμικού αριθμού γραμμικών προγραμμάτων, όπως θα δούμε δεν ισχύουν στην πράξη. Μιας και όπως θα δούμε στα πειράματα, τα βήματα είναι σταθερού αριθμού ακόμα κι αν μεγαλώνει πολύ το πρόβλημα και επίσης είναι πολύ λίγα σε σχέση με το μέγεθος του προβλήματος.

5.8.2 Ο Αλγόριθμος Γενικά

Ο αλγόριθμος στηρίζεται πάνω στην δημιουργία ενός ισοδύναμου προβλήματος βελτιστοποίησης (optimization) και στην επίλυση διάφορων γραμμικών προβλημάτων (LPs), με στόχο το να βρούμε κάποια κατά προσέγγιση ισορροπία Nash.

Η ιδέα όλη στηρίζεται πάνω στην δημιουργία μιας συνάρτησης [f(x,y)] που μετράει την μέγιστη απόκλιση των κερδών (payoffs) των δύο παικτών από το μέγιστο κέρδος (best payoffs) που θα μπορούσε να έχει κάθε παίκτης γνωρίζοντας την στρατηγική των άλλων παικτών.

Αυτής της συνάρτησης ψάχνουμε να βρούμε κάποιο stationary point της (πάνω στις κατανομές πιθανοτήτων των στρατηγικών των δύο παικτών x και y), στο οποίο αυτή ελαχιστοποιείται, ακολουθώντας πάντα μια εφικτή φθίνουσα (descent) κατεύθυνση της τιμής της συνάρτησης πάνω στις στρατηγικές των δύο παικτών.

Όταν μια τέτοια κατεύθυνση δεν υπάρχει [έχουμε φτάσει σε σημείο όπου η συνάρτηση f δεν μπορεί να πάρει χαμηλότερη τιμή], τότε έχουμε περιέλθει σε ένα σημείο (stationary point) όπου υπάρχει κατά προσέγγιση Ισορροπία Nash (approximate Nash Equilibrium).

Αυτό γίνεται μέσω μιας σαφής και άμεσα υλοποιήσιμης διαδικασίας που στο paper [1], ονομάζεται Descent Procedure. Μέσω αυτής ψάχνουμε εφικτές λύσεις που επιτυγχάνονται με την λύση κάποιων Γραμμικών Προβλημάτων (LPs) όπως θα δούμε στη συνέχεια.

Όπως μπορούμε να δούμε η όλη διαδικασία είναι εύκολα υλοποιήσιμη.
Για αυτόν τον λόγω χρησιμοποιώντας κώδικα σε ANSI C και έναν LP solver (CPLEX) για την λύση των γραμμικών προβλημάτων, επιτύχαμε εν τέλει την υλοποίηση του αλγορίθμου που περιγράφεται στο paper [1].

Αυτό έγινε βέβαια με κάποιες αλλαγές στον αλγόριθμο. Για αυτόν τον λόγο θα παρουσιάσουμε παρακάτω όλον τον αλγόριθμο υλοποίησης της Descent Procedure, όπως αυτή υλοποιήθηκε.

5.8.3 Σημειογραφία

Ο αλγόριθμος που περιγράφεται στο paper υλοποιήθηκε για δύο παίκτες, τον παίκτη γραμμής και στήλης.

Κάθε παίκτης έχει ένα σύνολο από στρατηγικές. Ο παίκτης γραμμής έχει σε σύνολο m (row_size) στρατηγικές και ο παίκτης στήλης έχει n (col_size) στρατηγικές. Έτσι καθορίζονται και οι πίνακες κέρδους (payoffs matrices) του κάθε παίκτη, μεγέθους m x n ο καθένας.

Ο πίνακας κέρδους (payoff matrix) του παίκτη γραμμής είναι ο R και του παίκτη στήλης είναι ο C.

Επίσης κάθε παίκτης καλείται να παίξει τις στρατηγικές του όπως αυτές καθορίζονται από τα διανύσματα πιθανοτικής κατανομής x (για τις στρατηγικές του παίκτη γραμμής) και y (για τις στρατηγικές του παίκτη στήλης).

Έτσι το πρόβλημα της εύρεσης ε – κατά προσέγγιση Ισορροπίας Nash σε ένα παίγνιο (R, C) για κάποιο $\varepsilon \geq 0$ εναπόκειται στο να βρεθούν οι πιθανοτικές κατανομές x και y πάνω στις στρατηγικές του κάθε παίκτη αντίστοιχα, έτσι ώστε να ικανοποιούνται οι εξής ανισότητες:

$$x^T R\bar{y} \leq \bar{x}^T R\bar{y} + \varepsilon \text{ , για κάθε πιθανοτική κατανομή x}$$

και

$$\bar{x}^T Cy \leq \bar{x}^T C\bar{y} + \varepsilon \text{ , για κάθε πιθανοτική κατανομή y}$$

Με άλλα λόγια κανείς παίκτης να μην μπορεί να αυξήσει το κέρδος του περισσότερο από ε, με μονόπλευρη αλλαγή στρατηγικής.

5.8.4 Συνάρτηση f(x,y)

Οι παραπάνω εξισώσεις υπολογισμού της ε – κατά προσέγγιση Ισορροπίας Nash, μπορούν να διαμορφωθούν ως εξής:

$$x^T R\bar{y} \le \bar{x}^T R\bar{y} + \varepsilon \Rightarrow \max(R\bar{y}) - \bar{x}^T R\bar{y} \le \varepsilon$$

γιατί το συμφέρον του παίκτη γραμμής είναι να παίξει την κατανομή στρατηγικών, εκείνη που του επιφέρει το μέγιστο κέρδος σε σχέση με την κατανομή στρατηγικών που έπαιξε ο παίκτης στήλης. Άρα ο παίκτης γραμμής καλείται ουσιαστικά να βρει την κατανομή πιθανοτήτων πάνω στις στρατηγικές του, που μεγιστοποιούν το $x^T R\bar{y}$. Αυτή η κατανομή προφανώς δίνει την μέγιστη τιμή του πίνακα $R\bar{y}$, δηλαδή το $\max(R\bar{y})$.

και

$$\bar{x}^T Cy \le \bar{x}^T C\bar{y} + \varepsilon \Rightarrow \max(Cy) - \bar{x}^T C\bar{y} \le \varepsilon$$

ομοίως για τον παίκτη στήλης.

Η συνάρτηση f(x,y) μετράει την απόκλιση των κερδών και των δύο παικτών από το max που μπορούν να φτάσουν.
Άρα για να βρω μια ε – κατά προσέγγιση Ισορροπίας Nash αρκεί να βρω τέτοια f(x,y) που να ισχύει:

$$\mathbf{f(x,y) \le \varepsilon}$$

Λόγω του ότι μπορεί ο κάθε παίκτης να έχει μικρότερη απόκλιση ε από το max σε σχέση με τον άλλο παίκτη, μετράμε πάντα ως ε την μέγιστη απόκλιση.

Τελικά η f(x,y) παίρνει την εξής μορφή:

$$f(x,y) = \max_{(x,y \in \Delta_m \Delta_n)} \{\max(Ry) - x^T Ry, \max(C^T x) - x^T Cy\}$$

Παρατήρηση: Η συνάρτηση f(x,y) δεν είναι ποτέ convex για τα x και y ταυτόχρονα.

5.8.5 Αναλυτική μεθοδολογία - Descent Procedure

5.8.5.1 Προεπεξεργασία

- Αρχικά καθορίζουμε τις τιμές των m (row_size) και n (col_size). Δηλαδή το πλήθος των στρατηγικών του παίκτη γραμμής και του παίκτη στήλης.

- Αφού γίνει αυτό, όπως είπαμε και στην περιγραφή των αρχείων εισόδου και εξόδου παραπάνω, ανοίγουμε όσα αρχεία θέλουμε να γράψουμε πληροφορίες σχετικά με μεγέθη που εμφανίζονται στο πρόγραμμα. Αυτά τα αρχεία τα γεμίζουμε κατά την διάρκεια του εκτέλεσης του προγράμματος.
  [*Δεν θα ασχοληθούμε με τον τρόπο που τα γεμίζουμε γιατί είναι προφανές πως και πότε γίνεται αυτό από την περιγραφή των αρχείων παραπάνω*]

- Στη συνέχεια δεσμεύουμε χώρο για όσους δυναμικούς πίνακες θα χρειαστούμε στο πρόγραμμά μας.

- Γίνεται ο καθορισμός των πινάκων κέρδους των δύο παικτών R και C.
  Οι τιμές των στοιχείων αυτών των πινάκων μπορούν να είναι οποιοσδήποτε δεκαδικός αριθμός double ακρίβειας.

  Σημείωση: Η δημιουργία αυτών των πινάκων θα αναλυθεί σε επόμενο κεφάλαιο, διότι αυτοί οι πίνακες καθορίζουν το είδος και την δυσκολία του εκάστοτε παιγνίου.

- Γίνεται κανονικοποίηση των τιμών των στοιχείων των δύο πινάκων σε τιμές μεταξύ 0 και 1.

- Καθορίζουμε τα αρχικά σημεία x και y.
  Μπορούμε να ξεκινήσουμε από οποιαδήποτε πιθανοτική κατανομή x και y.
  Επιλέγουμε να ξεκινήσουμε είτε από ομοιόμορφη κατανομή πάνω στα x και y [δηλαδή όλες οι τιμές του x ίσες με 1/m και όλες οι τιμές του y ίσες με 1/n], είτε από κάποια αγνή στρατηγική του κάθε παίκτη [δηλαδή όλες οι τιμές του x (ή y αντίστοιχα) ίσες με 0, εκτός από μία που παίρνει την τιμή 1]

5.8.5.2 Descent Procedure

Βήμα πρώτο [Step 1]

Αρχικά απελευθερώνουμε τα env και lp όπως είδαμε παραπάνω στην εξήγηση του πως δουλεύει η CPLEX.
Δημιουργούμε νέα env και lp για την δημιουργία ενός νέου Γραμμικού Προβλήματος (LP).

Κανονικοποιούμε τα διανύσματα πιθανοτήτων x και y.
[Βλέπε στην συνάρτηση **norm_prob_vector** παραπάνω, όπου περιγράφονται οι λόγοι για τους οποίους το κάνουμε αυτό].

Υπολογίζουμε την τιμή των $f_R(x,y)$ και $f_C(x,y)$, καθώς και της f(x,y) στο αρχικό σημείο (x, y) πριν εκτελεστεί το βήμα 1 του αλγορίθμου.
[Ας ονομάσουμε τις τιμές που παίρνουν αυτές οι συναρτήσεις f_C_val, f_R_val και f_val αντίστοιχα.]

Συγκρίνουμε τις τιμές που έχουν πάρει οι συναρτήσεις $f_R(x,y)$ και $f_C(x,y)$.

Αν $\mathbf{f_R(x,y) \geq f_C(x,y)}$ τότε καλείται η συνάρτηση populatebyrow ώστε να δημιουργήσουμε το LP που αντιστοιχεί στην συγκεκριμένη περίπτωση.

<u>populatebyrow – Start</u>

1. Καθορίζουμε τους δυναμικούς πίνακες που θα χρειαστούμε.

2. Καθορίζουμε το είδος του LP που θα λύσουμε.
   Στην περίπτωσή μας είναι πρόβλημα μεγιστοποίησης.

3. Καθορίζουμε την αντικειμενική συνάρτηση:
   Όπως βλέπουμε από τον αλγόριθμο καλούμαστε να ελαχιστοποιήσουμε την εξής συνάρτηση:
   $\max(Ry_0) - x^T Ry_0$
   με το y_0 σταθερό και μόνη μεταβλητή το x.

   Αντί αυτού αρκεί να μεγιστοποιήσουμε το $x^T Ry_0$.
   Αυτό ($x^T Ry_0$) αποτελεί και την αντικειμενική συνάρτηση του LP που καλούμαστε να λύσουμε.

4. Καθορίζουμε τους περιορισμούς του LP (m σε πλήθος):
   $$(C^T - e_n[Cy_0 + Ry_0]^T)x \le \max(Ry_0)e_n$$
   Αυτές αποτελούν m σε αριθμό περιορισμούς:
   $$(C^T - e_n[Cy_0 + Ry_0]^T)x_1 \le \max(Ry_0)e_n$$
   $$(C^T - e_n[Cy_0 + Ry_0]^T)x_2 \le \max(Ry_0)e_n$$
   $$\dots$$
   $$(C^T - e_n[Cy_0 + Ry_0]^T)x_m \le \max(Ry_0)e_n$$

   Υπάρχει ακόμα ένας περιορισμός:
   $$x_1 + x_2 + \dots + x_m = 1$$

populatebyrow – End

Αν $\mathbf{f_R(x,y) \le f_C(x,y)}$ τότε καλείται η συνάρτηση populatebyrow_2 ώστε να δημιουργήσουμε το LP που αντιστοιχεί στην συγκεκριμένη περίπτωση.

populatebyrow_2 – Start

1. Καθορίζουμε τους δυναμικούς πίνακες που θα χρειαστούμε.

2. Καθορίζουμε το είδος του LP που θα λύσουμε.
   Στην περίπτωσή μας είναι πρόβλημα μεγιστοποίησης.

3. Καθορίζουμε την αντικειμενική συνάρτηση:
   Όπως βλέπουμε από τον αλγόριθμο καλούμαστε να ελαχιστοποιήσουμε την εξής συνάρτηση:
   $\max(C^T x_0) - x_0{}^T Cy$
   με το x_0 σταθερό και μόνη μεταβλητή το y.

   Αντί αυτού αρκεί να μεγιστοποιήσουμε το $x_0{}^T Cy$.
   Αυτό ($x_0{}^T Cy$) αποτελεί και την αντικειμενική συνάρτηση του LP που καλούμαστε να λύσουμε.

4. Καθορίζουμε τους περιορισμούς του LP (n σε πλήθος):

$$(R - e_m[x_0{}^T R + x_0{}^T C])y \le \max(C^T x)e_m$$

Αυτές αποτελούν n σε αριθμό περιορισμούς:

$$(R - e_m[x_0{}^T R + x_0{}^T C])y_1 \le \max(C^T x)e_m$$
$$(R - e_m[x_0{}^T R + x_0{}^T C])y_2 \le \max(C^T x)e_m$$
$$\dots$$
$$(R - e_m[x_0{}^T R + x_0{}^T C])y_n \le \max(C^T x)e_m$$

Υπάρχει ακόμα ένας περιορισμός:

$$y_1 + y_2 + \dots + y_m = 1$$

populatebyrow 2 – End

Αν $\mathbf{f_R(x,y) = f_C(x,y)}$ τότε τα x και y δεν χρειάζονται καμία τροποποίηση και πηγαίνουμε κατ` ευθείαν στο βήμα 2 [Step 2].
Τα x και y μένουν ανεπηρέαστα.

Στη συνέχεια είμαστε έτοιμοι να λύσουμε το LP που έχουμε δημιουργήσει.
[*ο τρόπος που λύνεται μέσω της CPLEX αναφέρεται στο προηγούμενο κεφάλαιο*]

Η λύση θα μας δώσει τιμές στο διάνυσμα **x**, αν $f_R(x,y) \ge f_C(x,y)$ (και κρατάμε σταθερό το διάνυσμα y)
ή στο διάνυσμα **y**, αν $f_R(x,y) \le f_C(x,y)$ (και κρατάμε σταθερό το διάνυσμα x).

Μετά την λύση του LP έχουμε τα νέα διανύσματα πιθανοτικών κατανομών x και y [ας τα πούμε x_1 και y_1].

Αν υπολογίσουμε τις νέες τιμές των συναρτήσεων $f_R(x\_1,y\_1)$ και $f_C(x\_1,y\_1)$ με τα νέα διανύσματα x και y, θα διαπιστώσουμε ότι πλέον οι τιμές τους είναι ίσες.
Ή αν θέλουμε να το δούμε προσεγγιστικά όπως κάνουμε στο πρόγραμμα, βάσει της παραμέτρου δ, $| f_R(x\_1,y\_1) - f_C(x\_1,y\_1)| \le \delta$

Έτσι μετά το βήμα 1 υπολογίζουμε την νέα τιμή της συνάρτησης f(x_1,y_1)
[ας την ονομάσουμε f_val]

Κάνουμε τον έλεγχο να δούμε αν έχουμε φτάσει σε exact ισορροπία Nash.
Δηλαδή αν η νέα τιμή που έχουμε υπολογίσει $\mathbf{f(x\_1,y\_1) \le \delta}$ [αν δηλαδή η f που υπολογίσαμε είναι τυπικά κοντά στην τιμή που θεωρούμε 0]

- Αν **ναι** τότε έχουμε exact ισορροπία Nash και θέτουμε ένα flag [ας πούμε το flag **flag_ne**] ώστε να γνωρίζουμε ότι έχουμε φτάσει σε exact N.E. και προχωράμε κανονικά στο βήμα 2. Ακόμα κι αν έχουμε βρει ισορροπία εκτελούμε τον αλγόριθμο μέχρι το βήμα 4 γιατί μπορεί η τιμή να μειωθεί περαιτέρω [μιας και η σύγκριση έγινε προσεγγιστικά βάσει της παραμέτρου δ]

- Αν **όχι** τότε προχωράμε στο δεύτερο βήμα του αλγορίθμου.

Βήμα Δεύτερο [Step 2]

Απελευθερώνουμε τα env και lp όπως είδαμε παραπάνω στην εξήγηση του πως δουλεύει η CPLEX.
Δημιουργούμε νέα env και lp για την δημιουργία ενός νέου Γραμμικού Προβλήματος (LP).

Δημιουργούμε τον πίνακα **G(x_1, y_1)** μεγέθους (m+n) x (m+n) με τα νέα διανύσματα x και y που υπολογίσαμε στο βήμα 1.

$$G(x,y) = \begin{bmatrix} R - e_m x^T R & -e_m y^T R^T + e_m {e_m}^T x^T R y \\ -e_n x^T C + e_n {e_n}^T x^T C y & C^T - e_n x^T R \end{bmatrix}$$

Προχωράμε στην συνάρτηση populatebyrow_3 για να δημιουργήσουμε το νέο LP που έχουμε να λύσουμε στο βήμα 2.

populatebyrow_3 – Start
Στο δεύτερο βήμα έχουμε να λύσουμε ένα minimax πρόβλημα.
Το πρόβλημα αυτό με έναν τρόπο που θα δούμε αμέσως το μετατρέπουμε σε ένα πρόβλημα ελαχιστοποίησης (minimization)

1. Αρχικά υπολογίζουμε τα σύνολα $S_R(y) = \text{supp}\max(Ry)$ και
   $S_C(x) = \text{supp}\max(C^T x)$
   Τα σύνολα αυτά θα μας χρησιμέψουν στο να επιλέξουμε τους περιορισμούς όπως θα δούμε παρακάτω.

2. Καθορίζουμε την αντικειμενική συνάρτηση:
   Ως αντικειμενική συνάρτηση θα θέσουμε μια νέα μεταβλητή u.
   Λόγω όμως των περιορισμών που θα βάλουμε στην συνέχεια την διαμορφώνουμε ως εξής:
   $$0 * y'_1 + 0 * y'_2 + \ldots + 0 * y'_n + 0 * x'_1 + 0 * x'_2 + \ldots + 0 * x'_m + 1 * u$$
   με x` και y` νέες μεταβλητές.
   Και $0 \le y'_i \le 1$ και $0 \le x'_i \le 1$ για κάθε i

3. Καθορίζουμε τους περιορισμούς:
   Οι περιορισμοί καθορίζονται από τον παρακάτω πίνακα:
   $$G(x,y) * \begin{bmatrix} y` \\ x` \end{bmatrix} \le u * e_{m+n}$$
   δηλαδή αν το αναλύσουμε περαιτέρω:
   $(G_{11} * y'_1) + (G_{12} * y'_2) + \ldots + (G_{1(n+1)} * x'_1) + \ldots + (G_{1(n+m)} * x'_m) \le u$
   $(G_{21} * y'_1) + (G_{22} * y'_2) + \ldots + (G_{2(n+1)} * x'_1) + \ldots + (G_{2(n+m)} * x'_m) \le u$
   …
   $(G_{(m+n)1} * y'_1) + (G_{(m+n)2} * y'_2) + \ldots + (G_{(m+n)(n+1)} * x'_1) + \ldots + (G_{(m+n)(n+m)} * x'_m) \le u$

   όμως δεν παίρνουμε όλες τις γραμμές ως περιορισμούς.

Από τις πρώτες m γραμμές επιλέγουμε ως περιορισμούς αυτές που σηματοδοτούνται από το $S_R(y) = \text{supp}\max(Ry)$

Από τις υπόλοιπες n γραμμές επιλέγουμε ως περιορισμούς αυτές που σηματοδοτούνται από το $S_C(x) = \text{supp}\max(C^T x)$.

populatebyrow_3 – End

Αφού έχουμε καθορίσει το LP πλέον μπορούμε να το λύσουμε καλώντας την CPLEX όπως αναλύθηκε σε προηγούμενο κεφάλαιο.

Έτσι έχουμε τα παρακάτω αποτελέσματα:

- Η τιμή της αντικειμενική συνάρτησης u είναι η τιμή της V(x,y) [V(x,y) = u]
- Τα υπολογισμένα διανύσματα x` και y` (primal λύση) είναι τα νέα διανύσματα x και y.

Εκτός των **primal** λύσεων έχουμε και τις **dual** λύσεις. Κάθε μία αντιστοιχεί σε ένα από τους περιορισμούς που θέσαμε [και το πλήθος τους είναι όσο και το πλήθος των περιορισμών].
Αν **#S$_R$** είναι το πλήθος των στοιχείων του συνόλου $S_R(y)$, **#S$_C$** είναι το πλήθος των στοιχείων του συνόλου $S_C(x)$ και **dual_m** ο πίνακας που περιέχει κατά σειρά τις dual λύσεις του LP τότε:
Τα πρώτα #S_R στοιχεία του πίνακα dual_m διαιρεμένα με το άθροισμα των πρώτων #S_R στοιχείων αποτελούν τα στοιχεία του διανύσματος w, στις θέσεις όμως που το σύνολο S_R δεικτοδοτεί κατά σειρά. Στις υπόλοιπες θέσεις βάζουμε τιμή ίση με 0.
Έτσι που το διάνυσμα w να έχει τελικά μέγεθος ίσο με m.

Ομοίως τα υπόλοιπα #S_C στοιχεία του πίνακα dual_m διαιρεμένα με το άθροισμα των υπόλοιπων #S_C στοιχείων αποτελούν τα στοιχεία του διανύσματος z, στις θέσεις όμως που το σύνολο S_C δεικτοδοτεί κατά σειρά. Στις υπόλοιπες θέσεις βάζουμε τιμή ίση με 0.
Έτσι που το διάνυσμα z να έχει τελικά μέγεθος ίσο με n.

Τα **w** και **z** είναι οι dual λύσεις του προβλήματος που λύσαμε στο βήμα 2 του αλγορίθμου.

Έτσι προχωρούμε στον υπολογισμό του **λ** και του **μ**

Υπολογισμός λ [Step 2b]

Ο υπολογισμός του **λ** αποτελεί ένα ακόμη LP, πιο απλό, του οποίου την λύση αναζητούμε.

Απελευθερώνουμε τα env και lp όπως είδαμε παραπάνω στην εξήγηση του πως δουλεύει η CPLEX.
Δημιουργούμε νέα env και lp για την δημιουργία ενός νέου Γραμμικού Προβλήματος (LP).

Καλούμε την συνάρτηση populatebyrow_4 με ορίσματα το x_1 όπως αυτό καθορίστηκε μετά το βήμα 1 και το w όπως αυτό καθορίστηκε μετά το βήμα 2 (dual μεταβλητή)

populatebyrow_4 - Start

1. Καθορίζουμε το είδος του LP που θα λύσουμε.
   Στην περίπτωσή μας είναι πρόβλημα ελαχιστοποίησης (minimization).

2. Καθορίζουμε την αντικειμενική συνάρτηση το σύνολο:
   $$(w - x_1)^T Ry'$$
   με άγνωστη μεταβλητή την y'.
   Έτσι ορίζονται ουσιαστικά n στο σύνολο αντικειμενικές συναρτήσεις για το LP. Εμείς δεν θα χρησιμοποιήσουμε όλες τις γραμμές (n στο σύνολό τους) ως αντικειμενικές συναρτήσεις αλλά μόνο αυτές που δεικτοδοτούνται από το σύνολο $S_C(x_1)$

3. Καθορίζουμε τους περιορισμούς:
   Ο μόνος περιορισμός που θέτουμε είναι ο:
   $$y`_1 + y`_2 + ... + y`_m = 1$$
   λόγω του ότι το διάνυσμα y' πρέπει να είναι διάνυσμα πιθανοτήτων.

populatebyrow_4 - End

Αφού έχουμε καθορίσει το LP πλέον μπορούμε να το λύσουμε καλώντας την CPLEX όπως αναλύθηκε σε προηγούμενο κεφάλαιο.

Η τιμή του **λ** είναι η τιμή της αντικειμενικής συνάρτησης όπως αυτή επιστρέφεται από την λύση του LP.

Υπολογισμός μ [Step 2c]

Ο υπολογισμός του **μ** αποτελεί ένα ακόμη LP, πιο απλό, του οποίου την λύση αναζητούμε.

Απελευθερώνουμε τα env και lp όπως είδαμε παραπάνω στην εξήγηση του πως δουλεύει η CPLEX.
Δημιουργούμε νέα env και lp για την δημιουργία ενός νέου Γραμμικού Προβλήματος (LP).

Καλούμε την συνάρτηση populatebyrow_5 με ορίσματα το y_1 όπως αυτό καθορίστηκε μετά το βήμα 1 και το z όπως αυτό καθορίστηκε μετά το βήμα 2 (dual μεταβλητή)

populatebyrow_5 - Start

1. Καθορίζουμε το είδος του LP που θα λύσουμε.
   Στην περίπτωσή μας είναι πρόβλημα ελαχιστοποίησης (minimization).

2. Καθορίζουμε την αντικειμενική συνάρτηση το σύνολο:

$$x'^{T} C(z - y_1)$$

με άγνωστη μεταβλητή την x'.
Έτσι ορίζονται ουσιαστικά m στο σύνολο αντικειμενικές συναρτήσεις για το LP. Εμείς δεν θα χρησιμοποιήσουμε όλες τις γραμμές (m στο σύνολό τους) ως αντικειμενικές συναρτήσεις αλλά μόνο αυτές που δεικτοδοτούνται από το σύνολο $S_R(y_1)$

3. Καθορίζουμε τους περιορισμούς:
   Ο μόνος περιορισμός που θέτουμε είναι ο:
   $$x`_1 + x`_2 + ... + x`_m = 1$$
   λόγω του ότι το διάνυσμα x' πρέπει να είναι διάνυσμα πιθανοτήτων.

populatebyrow 5 - End

Αφού έχουμε καθορίσει το LP πλέον μπορούμε να το λύσουμε καλώντας την CPLEX όπως αναλύθηκε σε προηγούμενο κεφάλαιο.

Η τιμή του **μ** είναι η τιμή της αντικειμενικής συνάρτησης όπως αυτή επιστρέφεται από την λύση του LP.

Μετά τον υπολογισμό των **λ** και **μ** μπορούμε να προχωρήσουμε στο βήμα 4 της Descent Procedure.
ΠΡΟΣΟΧΗ: εκτελούμε το βήμα 4 πριν το βήμα 3 (κριτήρια τερματισμού) έτσι ώστε ακόμα κι αν είναι να τερματίσει το πρόγραμμα να εκτελέσει για τελευταία φορά τον υπολογισμό στο βήμα 4 ο οποίος μπορεί να ρίξει περαιτέρω την τιμή της συνάρτησης f(x,y)

Βήμα 4 [Step 4]

Στο βήμα αυτό προσπαθούμε να βρούμε ένα νέο σημείο (x,y) της συνάρτησης f(x,y) το οποίο να μειώνει την τιμή της συνάρτησης.

Αρχικά κανονικοποιούμε τα διανύσματα πιθανοτήτων x, y

Προχωρούμε στον υπολογισμό των του ε.
Ουσιαστικά στον υπολογισμό των ε* (όπως περιγράφεται στο Appendix A3 του paper [1]) και ε** (μια καλύτερη προσέγγιση του ε*)

Καλούμε την συνάρτηση epsilon_star η οποία υπολογίζει το ε*.

Epsilon star – Start

1. Καθορίζουμε τους δυναμικούς πίνακες που θα χρειαστούμε.

2. Υπολογίζουμε τα σύνολα $S_R(y) = \text{supp}\max(Ry)$ και $S_C(x) = \text{supp}\max(C^T x)$
   (μέσω της συνάρτησης suppmax_u)
   καθώς και τα συμπληρωματικά αυτών: $\overline{S}_R(y)$ και $\overline{S}_C(x)$ (μέσω της συνάρτησης

suppmax_u_bar)

3. Υπολογίζουμε τα:

$$\max(Ry)$$

καθώς επίσης και το

$$\max(Ry`),$$

για το δεύτερο, ελέγχοντας μόνο τις θέσεις που δεικτοδοτούνται από το σύνολο $S_R(y)$

4. Υπολογίζουμε το ε_1* ως εξής:

$$\varepsilon_1 * = \frac{\max(Ry) - (Ry)_i}{\max(Ry) - (Ry)_i + (Ry`)_i - \max_{S_R(y)}(Ry`)} \text{ για κάθε } i \in \overline{S}_R(y)$$

5. Υπολογίζουμε το ε_2* ως εξής:

$$\varepsilon_2 * = \frac{\max(C^T x) - (C^T x)_j}{\max(C^T x) - (C^T x)_j + (C^T x`)_j - \max_{S_C(x)}(C^T x`)} \text{ για κάθε } j \in \overline{S}_C(x)$$

Στην συνέχεια συγκρίνουμε τα ε_1*, ε_2* και 1 και επιστρέφουμε ως τιμή του ε* την μικρότερη από τις τρεις.

Epsilon_star – End

Στη συνέχεια προσπαθούμε να βελτιώσουμε την τιμή της ε* δημιουργώντας την ε** όπως περιγράφεται στην συνάρτηση epsilon_double_star.

epsilon_double_star – Start

1. Καθορίζουμε τους δυναμικούς πίνακες που θα χρειαστούμε.

2. Υποθέτοντας ότι μετά το βήμα 1 του αλγορίθμου οι τιμές των x και y είναι οι x, y και μετά το βήμα 2 οι νέες τιμές είναι οι x` και y` υπολογίζουμε τα παρακάτω:

$$H_R = (x`-x)^T R(y`-y) = (x`^T Ry` + x^T Ry) - (x`^T Ry + x^T Ry`)$$

$$H_C = (x`-x)^T C(y`-y) = (x`^T Cy` + x^T Cy) - (x`^T Cy + x^T Cy`)$$

και

$$\mathrm{H} = \min(H_R, H_C)$$

Αναλύουμε τις εκφράσεις περαιτέρω ώστε να αποφύγουμε κάποιες άσκοπες αφαιρέσεις που θα μπορούσαν αν επιφέρουν σφάλματα.

- Αν H≥0 τότε ε** = ε*
- Αν όχι τότε

$$\varepsilon ** = \min(\varepsilon *, \frac{|V(x,y) - f(x,y)|}{2 * |H|})$$

Η συνάρτηση τελικά επιστρέφει το ε**

epsilon_double_star – End

Η τιμή του **ε** (του βήματος που κάνουμε ώστε να προσεγγίσουμε καλύτερη τιμή της συνάρτησης f(x,y)) καθορίζεται ίση με την τιμή **ε****
[ε=ε**]

Στη συνέχεια υπολογίζουμε τη νέα τιμή της f.
Το νέο σημείο στο οποίο θα προχωρήσει η συνάρτηση είναι το:

$$x_3 = x + \varepsilon^{**}(x`-x) = (1-\varepsilon^{**})x + \varepsilon^{**}x`$$

$$y_3 = y + \varepsilon^{**}(y`-y) = (1-\varepsilon^{**})y + \varepsilon^{**}y`$$

Έτσι η νέα **f(x3, y3)** υπολογίζεται καλώντας την συνάρτηση f που υπολογίζει το εξής:

$$f_4 = f(x + \varepsilon^{**}(x`-x), y + \varepsilon^{**}(y`-y)) = f((1-\varepsilon^{**})x + \varepsilon^{**}x`, (1-\varepsilon^{**})y + \varepsilon^{**}y`)$$

Γίνεται κανονικοποίηση των διανυσμάτων πιθανοτήτων x_3, y_3

Γίνεται υπολογισμός της νέας τιμής της **f** βάσει μιας διαφορετικής **φόρμουλας** έτσι ώστε να γίνει έλεγχος της τιμής της f που υπολογίσαμε παραπάνω. Η φόρμουλα είναι η εξής:

$$f_{4f} = ((V(x,y) - f_1)\varepsilon^{**} - \varepsilon^{**2}\mathrm{H}) + f_1$$

Μετά από το βήμα 4 μπορούμε να πάμε στα κριτήρια τερματισμού όπως αυτά περιγράφονται στο βήμα 3, για να δούμε αν έχουμε φτάσει σε κάποιο stationary point (x,y)

Βήμα 3 [Step 3]

Στο βήμα αυτό δεν χρησιμοποιούνται όλα τα κριτήρια τερματισμού που αναφέρονται στο paper.

Ως **πρώτο κριτήριο** τερματισμού θέτουμε το εξής:

$$V(x,y)-f(x,y)\geq 0$$

Αλλά όπως είπαμε για λόγους προσεγγιστικούς χρησιμοποιούμαι και εδώ την παράμετρο δ και το κριτήριο διαμορφώνεται ως εξής:

$$V(x,y)-f(x,y)\geq -\delta$$

Αν ισχύει η παραπάνω ανισότητα τότε το πρόγραμμα έχει βρει ένα stationary point.

Σημείωση: Αντί αυτού του κριτηρίου τερματισμού μπορούμε να χρησιμοποιήσουμε ένα πιο χαλαρό κριτήριο που περιγράφεται από την παρακάτω ανισότητα:

$$f_4 \succ f_1(1-(\frac{\delta}{1+\delta})^2)$$

Ως **δεύτερο κριτήριο** χρησιμοποιούμαι τον έλεγχο αν μετά το βήμα 1 έχουμε βρει exact N.E. Αυτό το είχαμε ελέγξει στο βήμα 1 και αν συνέβη κάτι τέτοιο είχαμε θέσει ένα flag με όνομα **flag_ne**.

Αν **δεν** ισχύει κανένα από τα δύο κριτήρια θέτουμε ως αρχικά σημεία (x,y) τα (x_3, y_3) που βρήκαμε μετά το βήμα 4 και αρχίζουμε την διαδικασία από την αρχή (από το βήμα 1)

Αν ισχύει κάποιο από τα παραπάνω κριτήρια, σε κάθε μία από τις παραπάνω περιπτώσεις, το πρόγραμμα έχει φτάσει σε ένα σημείο στο οποίο η συνάρτηση δεν μπορεί να κατέβει χαμηλότερα. Ουσιαστικά έχουμε βρει μια κατά προσέγγιση ισορροπία Nash, η οποία όμως δεν είναι στα σημεία (x,y) που υπολογίστηκαν μετά το βήμα 1, αλλά τα σημεία **(x_3, y_3)** που υπολογίστηκαν στο βήμα 4. Και η τιμή της κατά προσέγγιση ισορροπίας Nash είναι η **f_4** που υπολογίσαμε στο τέλος του βήματος 4. Αυτά επιστρέφονται και ως έξοδο από το πρόγραμμα.

Έχουμε φτάσει λοιπόν σε ένα σημαντικό σημείο. Αν έχει τερματίσει ο αλγόριθμος τότε ελέγχουμε αν αυτό έγινε για πρώτη φορά. Θα ονομάσουμε το πρόβλημα που λύσαμε **primal πρόβλημα**.
Αν έγινε πρώτη φορά τερματισμός τότε παίρνουμε τις μεταβλητές **w** και **z** που είχαμε βρει στην τελευταία επανάληψη του αλγορίθμου (του primal προβλήματος) στο βήμα 2 και ξαναρχίζουμε την descent procedure από την αρχή μόνο που τώρα ως αρχικά σημεία x και y θέτουμε τα w και z αντίστοιχα [**x_0=w** και **y_0=z**]
Αυτό θα το ονομάσουμε **dual πρόβλημα**.

Εκτελούμε την ίδια ακριβώς διαδικασία που περιγράψαμε μέχρι να τερματίσει ξανά (για δεύτερη φορά). Το σημείο που τερμάτισε αποτελεί κι αυτό ένα stationary point της συνάρτησης, η οποία παίρνει μια νέα τιμή εδώ.

Συγκρίνουμε τις τιμές της f που έχουμε βρει κατά την λύση του **primal** προβλήματος, με αυτήν που έχουμε βρει κατά την λύση του **dual** προβλήματος και κρατάμε την μικρότερη.

Τέλος να πούμε ότι για να έχουμε exact N.E. πρέπει να ισχύει ότι η τιμή της **$f \leq \delta$.**

Σημείωση: Αφού τελειώσει το βήμα 3 κάνουμε έναν ακόμη έλεγχο για να δούμε αν η τιμή της συνάρτησης που υπολογίσαμε στο βήμα 4 είναι μεγαλύτερη της τιμής της συνάρτησης f που υπολογίσαμε μετά το βήμα 1. Αν κάτι τέτοιο συμβαίνει τότε έχουμε σίγουρα κάποιο σφάλμα στο αλγόριθμο που θα οφείλεται σε round off errors λόγω πράξεων μεταξύ αριθμών.
Μέχρι στιγμής δεν εμφανίστηκε κάποιο τέτοιο σφάλμα στις εκτελέσεις του κώδικα που έχουμε τρέξει.

Αφού τελειώσει η descent procedure απλώς αποδεσμεύουμε τον χώρο μνήμης που είχαμε δεσμεύσει για τους δυναμικούς πίνακες και τερματίσουμε το πρόγραμμα.

Το πρόγραμμα τερματίζει και τα επιθυμητά αποτελέσματα της εκτέλεσης αναφέρονται στα αρχεία .txt που αναφέραμε σε προηγούμενη ενότητα.

5.9 Ο Αλγόριθμος σχηματικά για τις μεταβλητές της CPLEX

Παρακάτω παρουσιάζεται σχηματικά ο αλγόριθμος της descent procedure περιγράψαμε παραπάνω και έχει να κάνει με διαμορφώνονται οι μεταβλητές της cplex (για παράδειγμα οι env και lp), όσο και οι μεταβλητές που εκφράζουν την συνάρτηση f, τα διανύσματα x και y, αλλά και μερικές που αποτελούν ενδιάμεσα αποτελέσματα.

while (1)
{

STEP_1:

Είσοδος: Διανύσματα πιθανοτήτων x, y

Πράξη: Ελευθερώνουμε τις μεταβλητές της CPLEX: lp, env και status

Δημιουργούμε νέες μεταβλητές για το νέο LP πρόβλημα:
env, lp, status

Καλούμε τη συνάρτηση populatebyrow (αν f_R > f_C) ή την populatebyrow_2 (αν f_R ≤ f_C) ώστε να δημιουργήσουμε το νέο LP με είσοδο τις πιθανοτικές κατανομές x και y

Έξοδος: Από την επίλυση του LP μέσω της CPLEX μας επιστρέφεται η x_cplex, που είναι διάνυσμα πιθανοτήτων.

Αν είχαμε καλέσει την populatebyrow τότε καταχωρούμε στο νέο διάνυσμα πιθανοτήτων x_1 το διάνυσμα x_cplex και στο y_1 το διάνυσμα εισόδου y.

Αν είχαμε καλέσει την populatebyrow_2 τότε καταχωρούμε στο νέο διάνυσμα πιθανοτήτων y_1 το διάνυσμα x_cplex και στο x_1 το διάνυσμα εισόδου x.

STEP_2:

Είσοδος: Διανύσματα πιθανοτήτων x_1, y_1

Πράξη: Υπολογισμός της τιμής της f(x_1,y_1)

Έλεγχος αν f_R≠f_C.
Αν όχι τότε έχει γίνει κάποιο σφάλμα στο βήμα 1

Έλεγχος αν μετά το βήμα έχει δημιουργηθεί exact N.E., δηλαδή f_val ≤ delta.
Αν ναι τότε προχωράμε μέχρι το τέλος του while και σταματάμε την διαδικασία

Ελευθερώνουμε τις μεταβλητές της CPLEX:

lp, env και status

Δημιουργούμε νέες μεταβλητές για το νέο LP πρόβλημα:
env, lp, status

Δημιουργούμε τον πίνακα G βάσει των διανυσμάτων πιθανότητας x_1 και y_1

Καλούμε τη συνάρτηση populatebyrow_3 ώστε να δημιουργήσουμε το νέο LP με είσοδο τις πιθανοτικές κατανομές x_1 και y_1

Έξοδος: Από την επίλυση του LP μέσω της CPLEX μας επιστρέφεται η x_cplex_2 και η y_cplex_2, που είναι διανύσματα πιθανοτήτων.

Επίσης η τιμή της αντικειμενικής συνάρτησης καταχωρείται στην V_x_y
και ως νέο διάνυσμα πιθανοτήτων x_2 το διάνυσμα x_cplex και y_2 το διάνυσμα εισόδου y_cplex_2.

Βάσει των dual μεταβλητών που επιστρέφονται από την λύση του LP δημιουργούμε τα ρ, (1-ρ), w και z

STEP_2b:

Είσοδος: Διανύσματα πιθανοτήτων x_1, y_1

Πράξη: Ελευθερώνουμε τις μεταβλητές της CPLEX:
lp, env και status

Δημιουργούμε νέες μεταβλητές για το νέο LP πρόβλημα:
env, lp, status

Καλούμε τη συνάρτηση populatebyrow_4 ώστε να δημιουργήσουμε το νέο LP με είσοδο τις πιθανοτικές κατανομές x_1 και w

Έξοδος: Από την επίλυση του LP μέσω της CPLEX μας επιστρέφεται η x_cplex_3 και η y_cplex_3, που είναι διανύσματα πιθανοτήτων.

Επίσης η τιμή της αντικειμενικής συνάρτησης καταχωρείται στην μεταβλητή lamda

STEP_2c:

Είσοδος: Διανύσματα πιθανοτήτων x_1, y_1

Πράξη: Ελευθερώνουμε τις μεταβλητές της CPLEX: lp, env και status

Δημιουργούμε νέες μεταβλητές για το νέο LP πρόβλημα:
env, lp, status

Καλούμε τη συνάρτηση populatebyrow_5 ώστε να δημιουργήσουμε το νέο LP με είσοδο τις πιθανοτικές κατανομές y_1 και z

Έξοδος: Από την επίλυση του LP μέσω της CPLEX μας επιστρέφεται η x_cplex_4 και η y_cplex_4, που είναι διανύσματα πιθανοτήτων.

Επίσης η τιμή της αντικειμενικής συνάρτησης καταχωρείται στην μεταβλητή mi

STEP_4:

Είσοδος: Διανύσματα πιθανοτήτων x_1, y_1, x_2, y_2 (normalized)

Πράξη: Υπολογισμός ε* βάσει των x_1, y_1, x_2, y_2

Υπολογισμός της f(x_1, y_1)

Υπολογισμός ε*

Υπολογισμός των νέων πιθανοτικών κατανομών x_3, y_3 (normalized)

Υπολογισμός της f4(x_3, y_3)

STEP_3:

Είσοδος: Διανύσματα πιθανοτήτων x_1, y_1, x_2, y_2 (normalized)

Πράξη:
```
if(V(x,y) – f ≥ -δ)
{
    if(flag_descent ≠ 1)
    {
        f_val_3 = f_val_4
        x = w
        y = z
        flag_descent = 1

        Ελευθερώνουμε τις μεταβλητές της
```

CPLEX:
lp, env και status

Μηδενισμός του μετρητή βημάτων

goto STEP_1
}

if(flag_descent ≠ 1)
{
Σύγκριση f_val_4 και f_val_3 και έξοδος του μικρότερου
}
}

CHECK:

Έλεγξε αν f_val_4 > f_val
Αν ναι τότε υπάρχει σφάλμα.

x = x_3
y = y_3

}

Κεφάλαιο 6

Εκτέλεση Προγράμματος - Πειράματα

Στο κεφάλαιο αυτό θα παρουσιαστεί αρχικά ο τρόπος εκτέλεσης του προγράμματος που υλοποιεί τον αλγόριθμο που μελετήσαμε.
Στη συνέχεια θα παρουσιαστούν κάποια πειράματα πάνω σε διαφορετικούς τύπους παιγνίων, τα οποία χαρακτηρίζονται από τους πίνακες κέρδους R και C του παίκτη γραμμής και στήλης αντίστοιχα, αλλά και από τις αρχικές πιθανοτικές κατανομές x και y. Μέσω τον πειραμάτων αυτών βγαίνουν σημαντικά αποτελέσματα όσο αφορά τον υπολογισμό ισορροπιών Nash και προσεγγιστικών ισορροπιών Nash..

6.1 Διαδικασία Εκτέλεσης Προγράμματος

Το πρόγραμμα υλοποιήθηκε και έτρεξε πάνω σε σύστημα UNIX και πιο συγκεκριμένα έτρεξε πάνω στον server zenon της σχολής Μηχανικών Η/Υ και Πληροφορικής.
Ο λόγος είναι ότι ο server αυτός περιέχει τόσο το πρόγραμμα CPLEX που χρησιμοποιούμε για την λύση των LPs (την βιβλιοθήκη που χρησιμοποιούμε στον κώδικα), όσο και τα licenses για να τρέξει η CPLEX.
Ο κώδικας είναι στο μεγαλύτερο του μέρος γραμμένος σε ANSI C και το compile γίνεται με gcc.
Παρακάτω θα περιγράψουμε αναλυτικά πως μπορεί κάποιος να εκτελέσει το πρόγραμμα.

6.1.1 Makefile

Έχουμε δημιουργήσει ένα makefile μέσω του οποίου γίνεται το compile πάνω στον server zenon. Αυτό γιατί χρειάζεται να γίνει include η επιπρόσθετη εξωτερική βιβλιοθήκη της CPLEX στον κώδικα.

ΠΡΟΣΟΧΗ: *Δεν έχει γίνει δοκιμή εκτέλεσης και compile σε άλλον server. Ενδεχομένως να υπάρχει πρόβλημα κατά την εκτέλεση του εκτελέσιμου σε διαφορετικό server, μιας και δεν έχει ελεγχθεί αν λειτουργεί σωστά ο license manager σε διαφορετικό σύστημα ή σε διαφορετική αρχιτεκτονική υπολογιστών.*

Για να γίνει σωστά το compile στο server zenon, αρκεί να αλλάξουμε μόνο το url EXSRC και να βάλουμε την διεύθυνση που υπάρχει το αρχείο **bsc_v3.c**, που είναι το αρχείο που υλοποιεί τον αλγόριθμο μας.

Αν θέλουμε να κάνουμε το compile σε διαφορετικό server, πρέπει να αλλάξουμε όλα τα links που αναφέρονται στο makefile, όπως αναφέρονται μέσα στο ίδιο το makefile αρχείο.

Αφού γίνει αυτό, τότε για να τρέξει το πρόγραμμα αρκεί να εκτελέσουμε δύο εντολές:

- **make clean**: Μέσω αυτής της εντολής σβήνονται τα αρχεία που ενδεχομένως να προέκυψαν από προηγούμενες εκτελέσεις του κώδικα.
- **make all**: Μέσω αυτής της εντολής γίνεται το compile του κώδικα (gcc) και η σύνδεση με όλες τις εξωτερικές βιβλιοθήκες που έχουμε κάνει include στο πρόγραμμά μας. Στην περίπτωσή μας υπάρχει μόνο μία εξωτερική βιβλιοθήκη, αυτή της CPLEX

ΠΡΟΣΟΧΗ: *Το αρχείο που κάνουμε compile και περιέχει τον κώδικα της υλοποίησης του αλγορίθμου έχει όνομα* ***bsc_v3.c*** *Αν θελήσουμε άλλο όνομα, τότε πρέπει να το αλλάξουμε και στο makefile και να το αντικαταστήσουμε σε όλα τα σημεία που αναφέρεται ως* ***bcs_v3***

Παρακάτω μπορούμε να δούμε τον κώδικα του makefile:

```
SYSTEM     = x86-64_rhel4.0_3.4
LIBFORMAT  = static_pic

#------------------------------------------------------------
#
# When you adapt this makefile to compile your CPLEX programs
# please copy this makefile and set CPLEXDIR and CONCERTDIR to
# the directories where CPLEX and CONCERT are installed.
#
#------------------------------------------------------------

CPLEXDIR      = /opt/cplex101
CONCERTDIR    = /opt/concert23
# ---------------------------------------------------------------------
# Compiler selection
# ---------------------------------------------------------------------

CC  = gcc

# ---------------------------------------------------------------------
# Compiler options
# ---------------------------------------------------------------------

CCOPT = -O -fPIC -fexceptions -DNDEBUG -DIL_STD
COPT  = -fPIC
JOPT  = -classpath /opt/cplex101/lib/cplex.jar -O

# ---------------------------------------------------------------------
# Link options and libraries
# ---------------------------------------------------------------------

CPLEXBINDIR   = $(CPLEXDIR)/bin/$(BINDIST)
CPLEXJARDIR   = $(CPLEXDIR)/lib/cplex.jar
CPLEXLIBDIR   = $(CPLEXDIR)/lib/$(SYSTEM)/$(LIBFORMAT)
CONCERTLIBDIR = $(CONCERTDIR)/lib/$(SYSTEM)/$(LIBFORMAT)

CCLNFLAGS = -L$(CPLEXLIBDIR) -lilocplex -lcplex -L$(CONCERTLIBDIR) -
lconcert -lm -lpthread
CLNFLAGS  = -L$(CPLEXLIBDIR) -lcplex -lm -lpthread
```

```
all:
	make all_c

execute: all
	make execute_c

CONCERTINCDIR = $(CONCERTDIR)/include
CPLEXINCDIR   = $(CPLEXDIR)/include

EXDIR         = $(CPLEXDIR)/examples
EXSRC         = /…
EXINC         = $(EXDIR)/include
EXDATA        = $(EXDIR)/data

CFLAGS  = $(COPT)  -I$(CPLEXINCDIR)
CCFLAGS = $(CCOPT) -I$(CPLEXINCDIR) -I$(CONCERTINCDIR)
JCFLAGS = $(JOPT)


#------------------------------------------------------------
#  make all      : to compile the examples.
#  make execute  : to compile and execute the examples.
#------------------------------------------------------------


C_EX = bcs_v3

all_c: $(C_EX)

execute_c: $(C_EX)
	 ./bcs_v3

# ------------------------------------------------------------

clean :
	/bin/rm -rf *.o *~ *.class
	/bin/rm -rf $(C_EX) $(CPP_EX)
	/bin/rm -rf *.mps *.ord *.sos *.lp *.sav *.net *.msg *.log *.clp

# ------------------------------------------------------------
#
# The examples
# ------------------------------------------------------------

bcs_v3: bcs_v3.o
	$(CC) $(CFLAGS) bcs_v3.o -o bcs_v3 $(CLNFLAGS)
bcs_v3.o: $(EXSRC)/bcs_v3.c
	$(CC) -c $(CFLAGS) $(EXSRC)/bcs_v3.c -o bcs_v3.o
```

6.1.2 Κυρίως Πρόγραμμα [bcs_v3.c]

Όπως είπαμε πιο πριν ο αλγόριθμός μας έχει υλοποιηθεί και περιλαμβάνεται στο αρχείο **bcs_v3.c**.
Σε αυτό δεν απαιτείται κάποια αλλαγή για να τρέξει το πρόγραμμα. Παρ` όλα αυτά πρέπει να γνωρίζουμε τα εξής πριν εκτελέσουμε τον κώδικα:

- Στον ίδιο φάκελο με το αρχείο, πρέπει να υπάρχει ένα txt αρχείο με όνομα **r_c_m.txt**
  Όπως είπαμε παραπάνω αυτό το αρχείο είναι αυτό που περιέχει τους πίνακες κέρδους R και C, που χαρακτηρίζουν το πρόγραμμα.
  Για την μορφή του αρχείου αυτού κοιτάξτε τις οδηγίες που δόθηκαν στο προηγούμενο Κεφάλαιο, όπου περιγράψαμε τα αρχεία Εισόδου/Εξοδου.
- Ο χρήστης μπορεί να επιλέξει τα **starting points x και y**. Αυτά μπορεί να είναι είτε uniform distributed, είτε να ξεκινάνε από κάποια τυχαία αγνή στρατηγική [από κάποια γωνία του πολυέδρου].
  Η επιλογή αφήνεται με ερώτημα στον χρήστη κατά την εκτέλεση του προγράμματος.
- Ο χρήστης πρέπει επίσης να επιλέξει **το μέγεθος του προβλήματος**. Δηλαδή τα μεγέθη **m** και **n**. Αυτά αποτελούν το πλήθος των στρατηγικών του παίκτη γραμμή και παίκτη στήλης αντίστοιχα.
  Ή διαφορετικά μπορούμε να πούμε ότι είναι το μέγεθος των πινάκων κέρδους R και C.
  Η επιλογή αφήνεται με ερώτημα στον χρήστη κατά την εκτέλεση του προγράμματος.

Μετά από την εισαγωγή των συγκεκριμένων αρχείων το πρόγραμμα εκτελείται και εμφανίζει τα αποτελέσματα όπως περιγράψαμε στο προηγούμενο κεφάλαιο.

6.2 Πειράματα

Παρακάτω περιγράφεται ένα πλήθος πειραμάτων που κάναμε για τον έλεγχο του προγράμματος. Σε κάθε πείραμα θα εξηγηθεί ο αλγόριθμος μέσω του οποίου υλοποιήθηκαν, θα δείξουμε τον κώδικα [στα πειράματα χρησιμοποιήθηκε είτε κώδικας σε ANSI C, είτε κώδικας σε MatLab] και κάποιες γραφικές παραστάσεις, όπως είδαμε παραπάνω, που δείχνουν τα ποιοτικά χαρακτηριστικά των πειραμάτων. Τέλος αναφέρονται τα συμπεράσματα των πειραμάτων ανάλογα με τα παίγνια τα οποία χρησιμοποιήθηκαν.

6.2.1 Γραφικές Παραστάσεις

Για το σχεδιασμό των γραφικών παραστάσεων χρησιμοποιήθηκε η Matlab 7.0 R14. Χρησιμοποιήθηκαν δύο είδη κώδικα, ανάλογα με το είδος των γραφικών παραστάσεων που θέλαμε.

Ο πρώτος κώδικας (**plots.m**) που χρησιμοποιήθηκε είναι ο εξής:

```
%The url of where we want to save the graph
src = "\…\name_of_graph";
%The url of where the info_graph_pr.txt file exists
M = dlmread('src\info_graph_pr.txt','\t');

m_1 = M(:,1);
m_1b = M(2:end,1);
m_2 = M(:,2);       % f1
m_3 = M(2:end,3);   % V
m_4 = M(2:end,4);   % e
m_5 = M(2:end,5);   % H
m_6 = M(:,6);       % f4

hold on

subplot(2,2,1)
plot(m_1, m_2, 'r*-', m_1, m_6, 'g+-', m_1b, m_3, 'kx-')
xlabel('# of steps')
%ylabel('f(x,y)')
title('Graph for the values of f_1(x,y), f_4(x,y) and  V')
legend('f1', 'f4', 'V', 1)

subplot(2,2,2)
plot(m_1b, m_5, '-o')
xlabel('# of steps')
ylabel('H')
title('Graph for the values of H')
legend('H', 1)

subplot(2,2,3)
plot(m_1b, m_4, '-o')
xlabel('# of steps')
ylabel('epsilon**')
```

```
title('Graph for the values of epsilon**')
legend('epsilon doulbe star', 2)
hold off

saveas(gcf, src,'jpg');
```

Επεξήγηση
Το πρόγραμμα αυτό διαβάζει τις πληροφορίες που έχουμε αποθηκεύσει για τον εκάστοτε αλγόριθμο στα αρχεία **info_graph_pr.txt** (για τον primal αλγόριθμο) και **info_graph_du.txt** (για τον dual αλγόριθμο).
Τα αρχεία αυτά περιέχουν 6 στήλες το καθένα.

- Η πρώτη στήλη παρουσιάζει τον αριθμό του βήματος (της επανάληψης) που βρίσκεται ο αλγόριθμος.
  ΠΡΟΣΟΧΗ: Όπου εμφανίζεται το βήμα -1 σημαίνει ότι αναφερόμαστε στο στάδιο που βρίσκεται το πρόγραμμα πριν εκτελέσει το πρώτο βήμα. Αναφερόμαστε δηλαδή στην αρχική κατάσταση.
- Η δεύτερη στήλη περιέχει την τιμή της συνάρτηση f(x,y), αφού εκτελεστεί το πρώτο βήμα του αλγορίθμου.
  ΠΡΟΣΟΧΗ: Στο βήμα -1 εμφανίζεται η τιμή της f πριν εκτελεστεί το βήμα 1, αρχικά.
- Η τρίτη στήλη περιέχει την τιμή της μεταβλητής V(x,y)
- Η τέταρτη στήλη περιέχει την τιμή της μεταβλητής ε**
- Η πέμπτη στήλη περιέχει την τιμή της μεταβλητής H
- Η έκτη στήλη περιέχει την τιμή της συνάρτηση f(x,y), αφού εκτελεστεί το τέταρτο βήμα του αλγορίθμου.
  ΠΡΟΣΟΧΗ: Στο βήμα -1 εμφανίζεται η τιμή της f πριν εκτελεστεί το βήμα 1, αρχικά.

Στη συνέχεια για κάθε παίγνιο που έχουμε για το οποίο έχουμε τρέξει το πρόγραμμα παράγει τρεις γραφικές παραστάσεις, που θα αναλύσουμε παρακάτω.

-o-

Ο δεύτερος κώδικας (**plots_2.m**) που χρησιμοποιήθηκε είναι ο εξής:

```
%The url of where we want to save the graph
src = "\…\name_of_graph";

%The url of where the info_graph_pr.txt file exists
M = dlmread('src\info_graph_pr.txt','\t');
m1_1 = M(:,1);
m1_1b = M(2:end,1);
m1_6 = M(2:end,6);      % f4

%The url of where the info_graph_pr.txt file exists
M = dlmread('src\info_graph_pr.txt','\t');
m2_1 = M(:,1);
m2_1b = M(2:end,1);
m2_6 = M(2:end,6);      % f4
```

```
%The url of where the info_graph_pr.txt file exists
M = dlmread('src\info_graph_pr.txt','\t');
m3_1 = M(:,1);
m3_1b = M(2:end,1);
m3_6 = M(2:end,6);      % f4

%The url of where the info_graph_pr.txt file exists
M = dlmread('src\info_graph_pr.txt','\t');
m4_1 = M(:,1);
m4_1b = M(2:end,1);
m4_6 = M(2:end,6);      % f4

%The url of where the info_graph_pr.txt file exists
M = dlmread('src\info_graph_pr.txt','\t');
m5_1 = M(:,1);
m5_1b = M(2:end,1);
m5_6 = M(2:end,6);      % f4

hold on

loglog(m1_1b, m1_6, 'g+-', m2_1b, m2_6, 'kx-', m3_1b, m3_6, '-o', m4_1b, m4_6, '--
o', m5_1b, m5_6, 'r*-')
xlabel('# of steps')
ylabel('f(x,y)')
title('Graph for the values of f(x,y)')
legend('Uniform', 'Pure strategy 1', 'Pure strategy 2','Pure strategy 3','Pure strategy 4',
1)

saveas(gcf, src,'jpg');
```

Επεξήγηση
Ο δεύτερος αυτός κώδικα είναι για να γίνουν γραφικές παραστάσεις της συνάρτησης f(x,y) μετά το βήμα 4, από διαφορετικά starting points x και y (1 από ομοιόμορφη και 4 από γωνίες του πολυέδρου) στην ίδια γραφική παράσταση.
Και αυτό το πρόγραμμα αυτό διαβάζει τις πληροφορίες που έχουμε αποθηκεύσει για τον εκάστοτε αλγόριθμο στα αρχεία **info_graph_pr.txt** (για τον primal αλγόριθμο) και **info_graph_du.txt** (για τον dual αλγόριθμο).
Απλώς χρησιμοποιεί μόνο την πρώτη και την τελευταία στήλη των αρχείων αυτών

6.2.2 Επεξήγηση γραφικών

Έγινε ένα πλήθος πειραμάτων πάνω σε διαφορετικά είδη παιγνίων και πάνω σε διαφορετικά μεγέθη στρατηγικών m και n, για κάθε παίγνιο.

Γενικά
Για κάθε παίγνιο θα παραθέτουμε τόσο τον αλγόριθμο με τον οποίο δημιουργήθηκαν, όσο και στοιχεία που το διαχωρίζουν από τα άλλα παίγνια, δηλαδή το μέγεθος του παιγνίου, τα starting points x και y και το stopping κριτήριο που χρησιμοποιήθηκε.

Στη συνέχεια θα παραθέτουμε κάποιες γραφικές που δημιουργήθηκαν για το εκάστοτε πρόβλημα μέσω των προγραμμάτων που αναφέραμε παραπάνω.
Για κάθε γραφική παράσταση και κάθε παίγνιο θα αναφέρεται η διεύθυνση του info.txt αρχείου που περιέχει περαιτέρω αποτελέσματα για κάθε παίγνιο, όπως περιγράψαμε στην ανάλυση των αρχείων εισόδου/εξόδου παραπάνω.

Στο τέλος θα παρουσιάζονται κάποια συμπεράσματα που αφορούν είτε συνολικά τα παίγνια, είτε κάθε είδος παιγνίου ξεχωριστά

Γραφικές Παραστάσεις μέσω του πρώτου κώδικα
Για κάθε παίγνιο θα παρουσιάσουμε ένα set τριών γραφικών για την descent διαδικασία που αρχίζει είτε με uniform κατανομές των διανυσμάτων πιθανοτήτων x και y, είτε από κάποια αγνή στρατηγική για τους δύο παίκτες, μέχρι να σταματήσει ο αλγόριθμος (θα τον ονομάσουμε primal αλγόριθμο) και ένα set τριών γραφικών παραστάσεων για την descent διαδικασία που ξεκινάει με αρχικά x και y τα διανύσματα πιθανοτήτων w και z, που βρήκαμε από την λύση του dual προβλήματος στην τελευταία επανάληψη του αλγορίθμου κατά την πρώτη descent διαδικασία (θα τον ονομάσουμε dual αλγόριθμο).

- Στο set των τριών γραφικών, η πάνω αριστερά γραφική παράσταση παρουσιάζει τις τιμές της συνάρτησης f μετά το πέρας του πρώτου βήματος (f1), μετά το πέρας του βήματος 4 (f4), καθώς και την τιμή της μεταβλητής V σε συνάρτηση των βημάτων που έτρεξε ο αλγόριθμος.
  Η μεταβλητή V εκφράζει το μέτρο αντίστασης της ισορροπίας (threat point).

- Η πάνω δεξιά γραφική παράσταση παρουσιάζει την τιμή της μεταβλητής H, που υπολογίστηκε κατά τον υπολογισμό της ε**, συναρτήσει των βημάτων που έτρεξε ο αλγόριθμος.

- Τέλος, η κάτω δεξιά γραφική παράσταση παρουσιάζει τις τιμές που παίρνει η μεταβλητή ε** συναρτήσει των βημάτων που έτρεξε ο αλγόριθμος.

Γραφικές Παραστάσεις μέσω του δεύτερου κώδικα
Για μερικά παίγνια θα παρουσιάσουμε επίσης μία γραφική που παρουσιάζει μόνο την τιμή που παίρνει η συνάρτηση f(x,y), αλλά αρχίζοντας από διαφορετικά starting points x και y.
Το πρώτο starting point θα είναι ομοιόμορφη κατανομή πάνω στα σύνολα x και y.
Τα επόμενα τέσσερα starting points θα είναι αγνές στρατηγικές. Δηλαδή όλα τα στοιχεία του x και y θα είναι 0 και μόνο ένα θα έχει την τιμή 1 στο x και y αντίστοιχα. Το σημείο που επιλέγουμε να βάλουμε 1 στα x και y είναι σε αυτό που η συνάρτηση f(x,y) παίρνει αρχική τιμή ίση με 1. Αυτά τα σημεία είναι εκείνα στα οποία είτε ο πίνακας R, είτε ο πίνακας C έχει τιμή 0 (ή έστω μια τιμή κοντά στο 0).
Στην γραφική παρουσιάζονται και οι 5 καμπύλες τα διαφόρων σημείων αυτών.
Σε λεζάντα παρουσιάζεται και η αρχική τιμή της f(x,y) για κάθε παράδειγμα.

Σημείωση: Η ακρίβεια με την οποία κάνουμε όλους τους υπολογισμός ώστε να αποφύγουμε αριθμητικά σφάλματα (κυρίως round off errors) είναι 10^{-3} ($\delta = 0.001$)

6.2.3 Δύσκολα Παίγνια

Κατά την εκπόνηση των πειραμάτων θα προσπαθήσουμε να δημιουργήσουμε δύσκολα παίγνια. Δηλαδή παίγνια τα οποία να μην δίνουν εύκολα ακριβείς ισορροπίες Nash (exact Nash Equilibria).
Έτσι δημιουργήσαμε παίγνια τα οποία να μην συγκλίνουν εύκολα σε:

- **Ισορροπίες με αγνές στρατηγικές (pure strategy equilibria)**
  Δηλαδή ισορροπίες στις οποίες ο παίχτης γραμμής να παίζει μόνο μία από τα σύνολο των στρατηγικών του (μία από τις m γραμμές) και ο παίχτης στήλης να παίζει μόνο μία από τα σύνολο των στρατηγικών του (μία από τις n στήλες)
- **2x2 μεγέθους ισορροπίες**
  Δηλαδή ισορροπίες που οι δύο παίκτες χρησιμοποιούν μονάχα δύο αγνές στρατηγικές ο καθένας, αντίστοιχα.
- **Ισορροπίες με ομοιόμορφη κατανομή πάνω στις στρατηγικές**
  Δηλαδή στρατηγικές όπου ο κάθε παίκτης παίζει ομοιόμορφα όλες τις αγνέ στρατηγικές που έχει στην διάθεσή του.

Επίσης προσπαθήσαμε να αποφύγουμε:

- **Πίνακες κέρδους (R και C) που να οδηγούν σε εύκολα constant sum παίγνια.**
  Constant sum παίγνια λέμε αυτά που το άθροισμα του κέρδους όλων των παικτών είναι πάντα σταθερό. Αυτά τα παίγνια όταν κανονικοποιούνται μετατρέπονται σε zero sum παίγνια, που θεωρούνται εύκολα.

Στα παίγνια που θα δημιουργήσουμε:

- Δοκιμάσαμε μεγάλο πλήθος παιγνίων. Από αυτά κρατήσαμε τα πιο χαρακτηριστικά
- Δοκιμάστηκαν παίγνια μεγέθους από 10x10 μέχρι 100x100
  [*Ερευνητικά δοκιμάστηκαν και σε μεγαλύτερα παίγνια όπως 200x200 ή 500x500, αλλά η συμπεριφορά τους ήταν παρόμοια με αυτήν των 100x100, οπότε δεν θα παρουσιάσουμε τέτοιου είδους πειράματα*]
- Ως αρχικές πιθανοτικές κατανομές x και y (starting points) θα χρησιμοποιήσουμε τέτοιες ώστε να αρχίζουμε είτε από ομοιόμορφη κατανομή (uniform distribution), είτε τέτοιες ώστε να ξεκινάμε από κάποια γωνία του πολυέδρου, δηλαδή από κάποια αγνή στρατηγική του παίκτη γραμμής και στήλης αντίστοιχα.
  Για να επιλέξουμε κάποια αγνή στρατηγική κάποιου παίκτη, αρκεί να θέσουμε όλες τις θέσεις του διανύσματος x για τον παίκτη γραμμής και του διανύσματος y για τον παίκτη στήλης, ίσο με 0, εκτός από μια θέση, στα διανύσματα x και y αντίστοιχα, που θέτουμε ίση με 1 και σηματοδοτεί την αγνή στρατηγική που θα παίξει ο εκάστοτε παίκτης.
- Το δ που σηματοδοτεί την ακρίβεια θα είναι πάντα 10^{-3}
- Stopping κριτήριο:
  Θα χρησιμοποιηθεί είτε το πρώτο ($V(x,y)-f(x,y)\geq -\delta$), είτε το δεύτερο
  $(f_4 \succ f_1(1-(\frac{\delta}{1+\delta})^2))$.
  Το δεύτερο κριτήριο τερματισμού θα χρησιμοποιηθεί μόνο εκεί όπου έχουμε δύσκολα παίγνια και χρειάζεται μεγαλύτερη χαλαρότητα ώστε να φτάσει σε μικρότερη τιμή προσεγγιστικής ισορροπίας Nash.

6.3 Τυχαίοι πίνακες κέρδους R και C (Random Matrices R and C)

Το πρώτο παίγνιο που δημιουργούμε είναι αυτό των τυχαίων πινάκων κέρδους R και C. Δηλαδή πινάκων που περιέχουν ως κέρδη τυχαίους δεκαδικούς αριθμούς double ακρίβειας.

Χαρακτηριστικά παιγνίων:

- Μέσω των τυχαίων πινάκων κέρδους προσπαθούμε να αποφύγουμε πίνακες που είτε προσεγγίζουν constant sum παίγνια, είτε έχουν μικρά supports, κάτι το οποίο δεν θέλουμε.
- Παρ` όλα αυτά τα παίγνια αυτά δεν θεωρούνται αρκετά δύσκολα.
  Όμως πρέπει να τα μελετήσουμε διότι αποτελούν ένα καλό μέτρο για την συμπεριφορά του αλγορίθμου και του προγράμματος σε τυχαία παίγνια. Τα αποτελέσματα όπως θα δούμε παρακάτω δείχνουν καθαρά ότι ο αλγόριθμος έχει μια άριστη μέση συμπεριφορά σε τυχαίες περιπτώσεις παιγνίων.
- Οι πίνακες R και C θα είναι τυχαίοι πίνακες, των οποίων οι τιμές (payoffs values) θα κανονικοποιούνται μεταξύ 0 και 1.

Οι τυχαίοι πίνακες R και C παράγονται μέσω κώδικα ANSI C μέσα στο κυρίως πρόγραμμα.
Ο κώδικας είναι ο παρακάτω:

```
/* Make random R, C matrices */
// Initialize random generator
srand(time(NULL));

/* Generate random numbers */
for(i=0 ; i<m ;i++)
{
        for(j=0 ; j<n ; j++)
        {
                R_matrix_fixed[i][j] = (double)rand();
                C_matrix_fixed[i][j] = (double)rand();
        } //end for loop
} //end for loop
```

Παρακάτω παρουσιάζονται μερικά χαρακτηριστικά παραδείγματα.

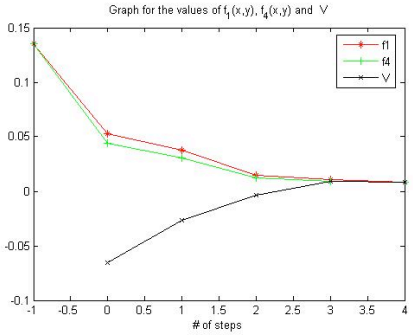
6.3.1 Τυχαίοι R και C πίνακες κέρδους μεγέθους 10 x 10

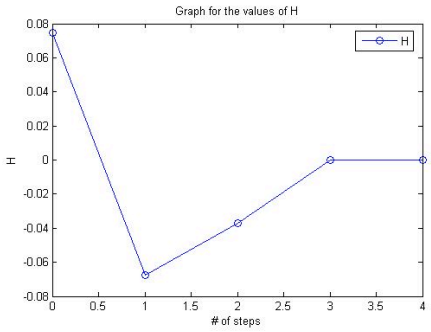
Χαρακτηριστικά Παιγνίου:

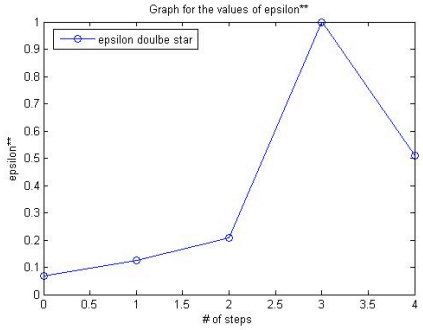
- Τύπος Παιγνίου: Τυχαίοι πίνακες κέρδους R και C
- Μέγεθος παιγνίου: 10 x 10
- Κριτήριο τερματισμού: $V(x,y)-f(x,y)\geq -\delta$
- Τιμή της παραμέτρου δ: 10^{-3}
- Αρχικές τιμές των x και y: Ομοιόμορφη κατανομή

Παράδειγμα 1 [αρχείο: *random\10x10\ \1*]

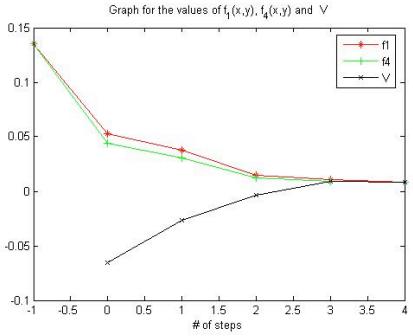
Primal Graph

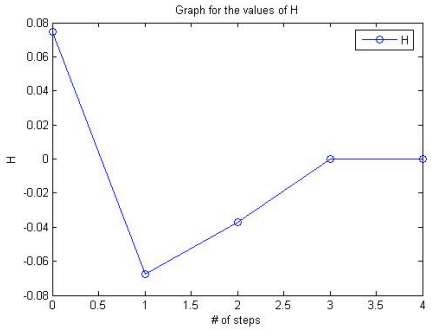


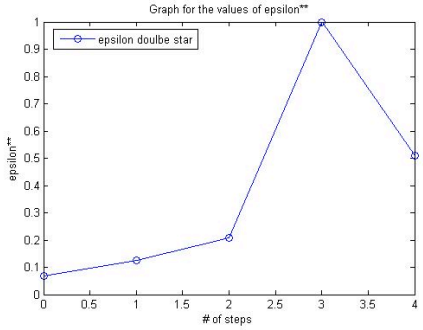




Dual Graph







Πίνακες Αποτελεσμάτων Συνολικών Πειραμάτων

Random Payoff Matrices R and C Primal Problem					
	Παράδειγμα 1	**Παράδειγμα 2**	**Παράδειγμα 3**	**Παράδειγμα 4**	**Παράδειγμα 5**
Ισορροπία Nash	0.008403	0	0.002218	0.002566	0.002698
Αρχική τιμή της f(x,y)	0.135402	0.164529	0.116814	0.155039	0.209373
Αριθμός Βημάτων	5	4	3	10	8
Αρχικά x και y	uniform	uniform	uniform	uniform	uniform

Random Payoff Matrices R and C Dual Problem					
	Παράδειγμα 1	**Παράδειγμα 2**	**Παράδειγμα 3**	**Παράδειγμα 4**	**Παράδειγμα 5**
Ισορροπία Nash	0.00926	0.001647	0	0	0.002698
Αρχική τιμή της f(x,y)	0.243494	0.025507	0.046627	0.250512	0.047519
Αριθμός Βημάτων	7	3	6	4	6

Σημείωση: Οι αντίστοιχες γραφικές παραστάσεις και τα αρχεία εξόδου για τα υπόλοιπα παραδείγματα φαίνονται στα αρχεία κάτω από τον φάκελο \random\10x10

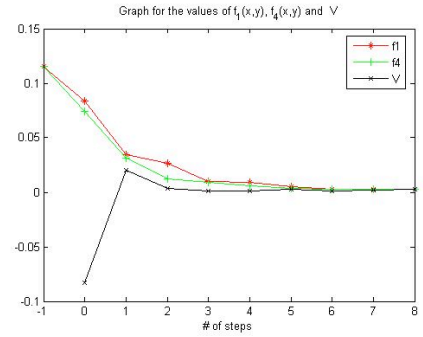
6.3.2 R και C πίνακες μεγέθους 20 x 20

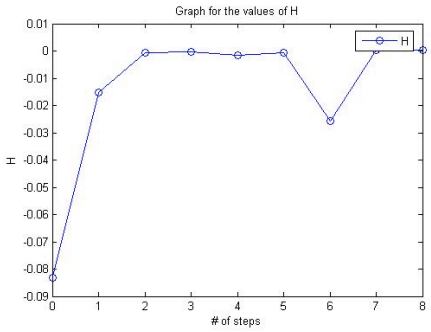
Χαρακτηριστικά Παιγνίου:

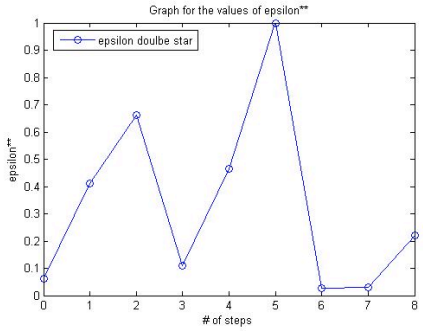
- Τύπος Παιγνίου: Τυχαίοι πίνακες κέρδους R και C
- Μέγεθος παιγνίου: 20 x 20
- Κριτήριο τερματισμού: $V(x,y)-f(x,y)\geq -\delta$
- Τιμή της παραμέτρου δ: 10^{-3}
- Αρχικές τιμές των x και y: Ομοιόμορφη κατανομή

Παράδειγμα 1 [αρχείο: *random\20x20 \1*]

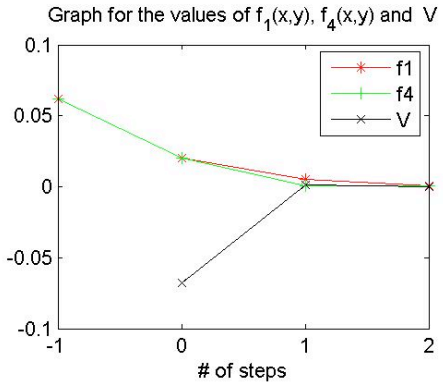
Primal Graph

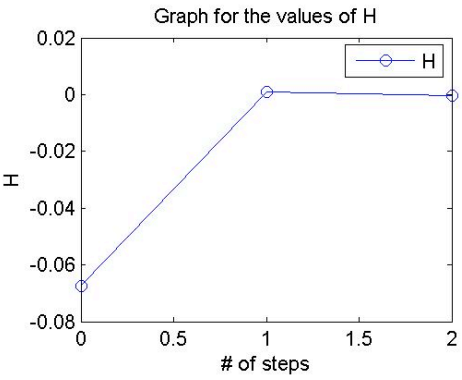


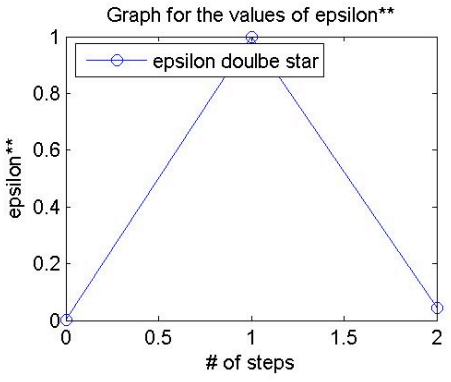




Dual Graph







Πίνακες Αποτελεσμάτων Συνολικών Πειραμάτων

Random Payoff Matrices R and C Primal Problem					
	Παράδειγμα 1	**Παράδειγμα 2**	**Παράδειγμα 3**	**Παράδειγμα 4**	**Παράδειγμα 5**
Ισορροπία Nash	0.00278	0.001018	0.004538	0.000275	0.001106
Αρχική τιμή της f(x,y)	0.115406	0.092297	0.121875	0.175771	0.170312
Αριθμός Βημάτων	9	5	10	8	8
Αρχικά x και y	uniform	uniform	uniform	uniform	uniform

Random Payoff Matrices R and C Dual Problem					
	Παράδειγμα 1	**Παράδειγμα 2**	**Παράδειγμα 3**	**Παράδειγμα 4**	**Παράδειγμα 5**
Ισορροπία Nash	0.000456	0.001294	0.00169	0.00166	0.0003
Αρχική τιμή της f(x,y)	0.061837	0.120407	0.140682	0.018441	0.063856
Αριθμός Βημάτων	3	7	6	4	8

Σημείωση: Οι αντίστοιχες γραφικές παραστάσεις και τα αρχεία εξόδου για τα υπόλοιπα παραδείγματα φαίνονται στα αρχεία κάτω από τον φάκελο \random\20x20

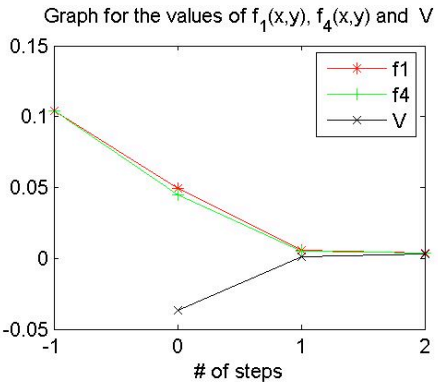
6.3.3 R και C πίνακες μεγέθους 30 x 30

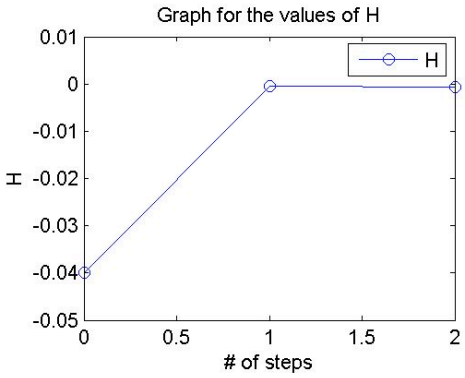
Χαρακτηριστικά Παιγνίου:

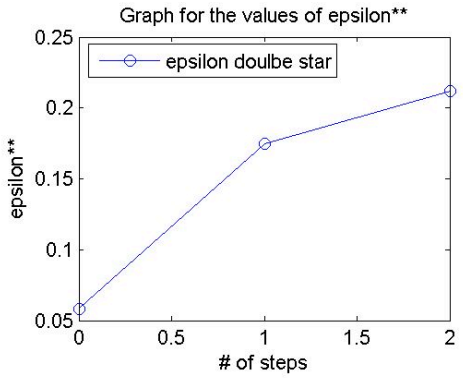
- Τύπος Παιγνίου: Τυχαίοι πίνακες κέρδους R και C
- Μέγεθος παιγνίου: 30 x 30
- Κριτήριο τερματισμού: $V(x,y)-f(x,y)\geq -\delta$
- Τιμή της παραμέτρου δ: 10^{-3}
- Αρχικές τιμές των x και y: Ομοιόμορφη κατανομή

Παράδειγμα 1 [αρχείο: *random\30x30 \1*]

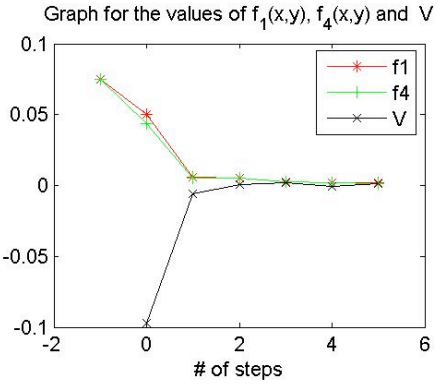
Primal Graph

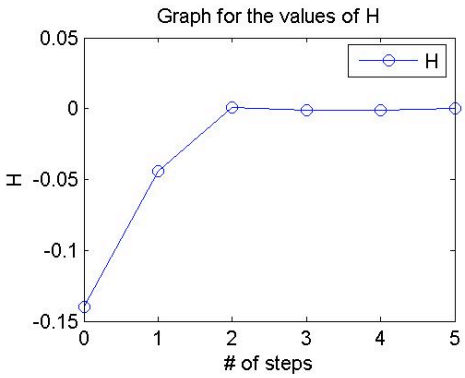


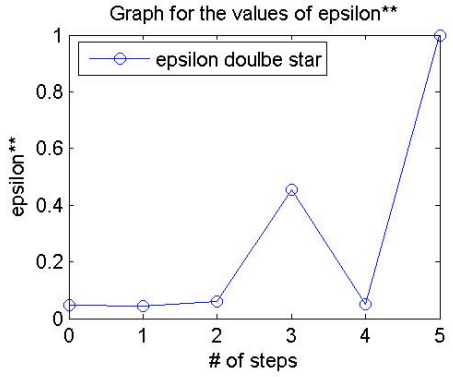




Dual Graph







Πίνακες Αποτελεσμάτων Συνολικών Πειραμάτων

Random Payoff Matrices R and C Primal Problem					
	Παράδειγμα 1	**Παράδειγμα 2**	**Παράδειγμα 3**	**Παράδειγμα 4**	**Παράδειγμα 5**
Ισορροπία Nash	0.003558	0.004029	0.000404	0.003360	0.000602
Αρχική τιμή της f(x,y)	0.104167	0.094005	0.083474	0.145694	0.132097
Αριθμός Βημάτων	3	9	5	5	4
Αρχικά x και y	uniform	uniform	uniform	uniform	uniform

Random Payoff Matrices R and C Dual Problem					
	Παράδειγμα 1	**Παράδειγμα 2**	**Παράδειγμα 3**	**Παράδειγμα 4**	**Παράδειγμα 5**
Ισορροπία Nash	0.00110	0.004596	0.002191	0.000089	0.000397
Αρχική τιμή της f(x,y)	0.074935	0.093015	0.035320	0.064478	0.106941
Αριθμός Βημάτων	6	8	5	3	3

Σημείωση: Οι αντίστοιχες γραφικές παραστάσεις και τα αρχεία εξόδου για τα υπόλοιπα παραδείγματα φαίνονται στα αρχεία κάτω από τον φάκελο \random\30x30

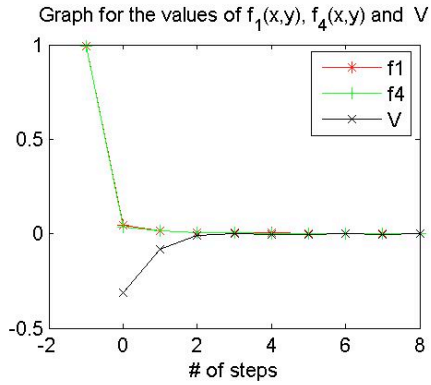
6.3.4 R και C πίνακες μεγέθους 100 x 100

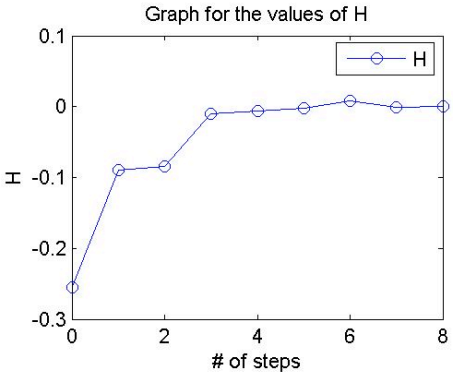
Χαρακτηριστικά Παιγνίου:

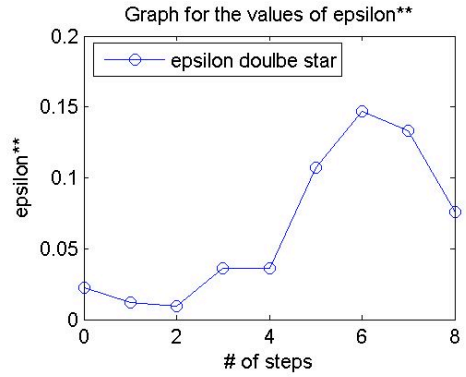
- Τύπος Παιγνίου: Τυχαίοι πίνακες κέρδους R και C
- Μέγεθος παιγνίου: 100 x 100
- Κριτήριο τερματισμού: $V(x,y)-f(x,y)\geq -\delta$
- Τιμή της παραμέτρου δ: 10^{-3}
- Αρχικές τιμές των x και y: Αγνές στρατηγικές

Παράδειγμα 1 [αρχείο: *random\100x100 \1*]

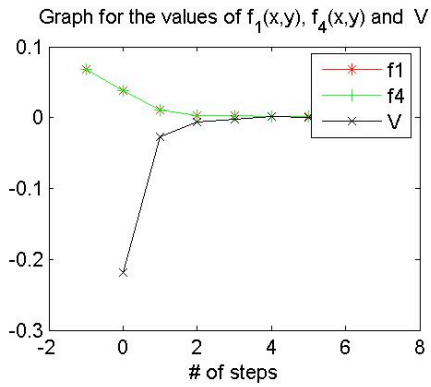
Primal Graph

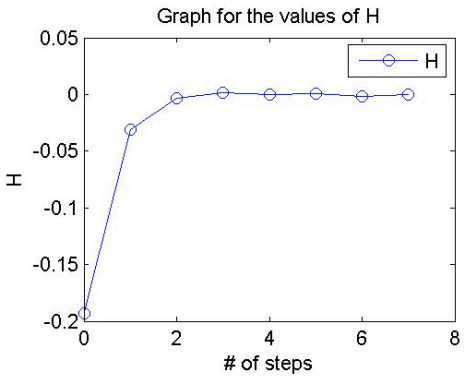


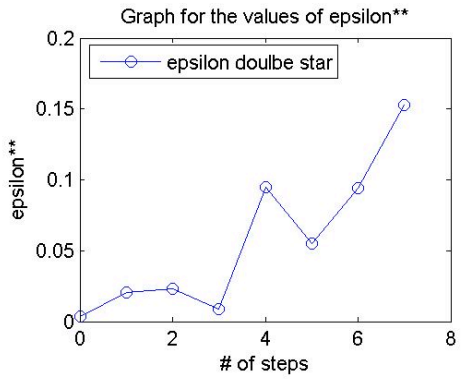




Dual Graph







Πίνακες Αποτελεσμάτων Συνολικών Πειραμάτων

Random Payoff Matrices R and C **Primal Problem**					
	Παράδειγμα 1	**Παράδειγμα 2**	**Παράδειγμα 3**		
Ισορροπία Nash	0.000642	0.000638	0.001615		
Αρχική τιμή της f(x,y)	0.993918	0.997252	0.989347		
Αριθμός Βημάτων	9	10	3		
Αρχικά x και y	Pure strategy on x and y	Pure strategy on x and y	Pure strategy on x and y		

Random Payoff Matrices R and C **Dual Problem**					
	Παράδειγμα 1	**Παράδειγμα 2**	**Παράδειγμα 3**		
Ισορροπία Nash	0.00095	0.00037	0.00061		
Αρχική	0.068174	0.096142	0.041649		

τιμή της f(x,y)					
Αριθμός Βημάτων	8	7	5		

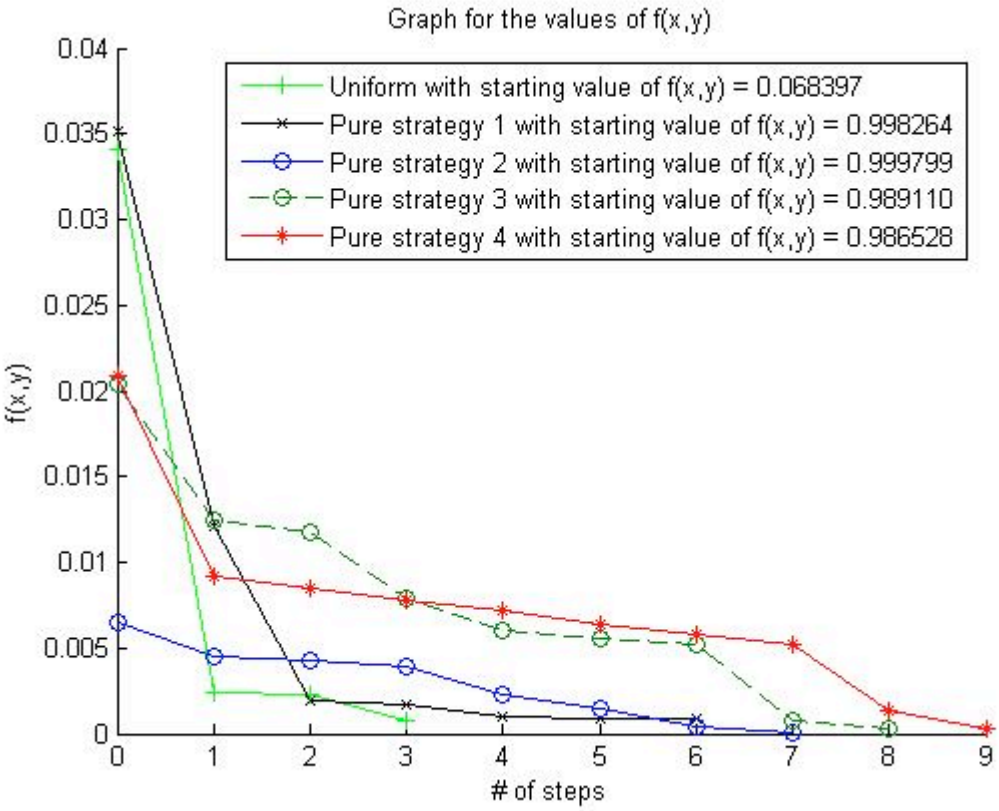
Σημείωση: Οι αντίστοιχες γραφικές παραστάσεις και τα αρχεία εξόδου για τα υπόλοιπα παραδείγματα φαίνονται στα αρχεία κάτω από τον φάκελο \random\100x100

6.3.5 R και C πίνακες μεγέθους 100 x 100

Χαρακτηριστικά Παιγνίου:

- Τύπος Παιγνίου: Τυχαίοι πίνακες κέρδους R και C
- Μέγεθος παιγνίου: 100 x 100
- Κριτήριο τερματισμού: $f_4 \succ f_1(1-(\frac{\delta}{1+\delta})^2)$
- Τιμή της παραμέτρου δ: 10^{-3}
- Αρχικές τιμές των x και y: Ομοιόμορφη κατανομή και αγνές στρατηγικές

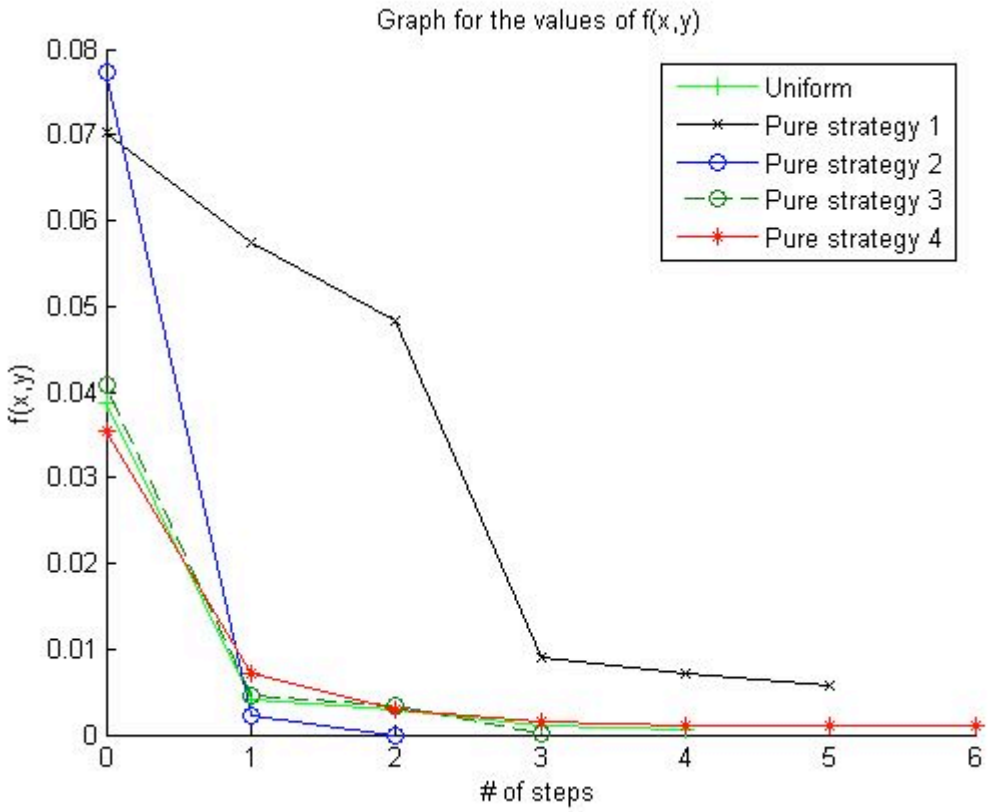
Primal Graph



Πίνακας Αποτελεσμάτων Πειράματος

Random Payoff Matrices R and C **Primal Problem**					
	Παράδειγμα 1	**Παράδειγμα 2**	**Παράδειγμα 3**	**Παράδειγμα 4**	**Παράδειγμα 5**
N.E.	0.000789	0.000854	0.000009	0.000284	0.000245
Αρχική τιμή της f	0.068397	0.998264	0.999799	0.989110	0.986528
Αριθμός Βημάτων	4	7	8	9	10
Αρχικά x και y	Uniform	Pure strategy on x and y	Pure strategy on x and y	Αρχικά x και y	Αρχικά x και y

Dual Graph



Πίνακας Αποτελεσμάτων Πειράματος

Random Payoff Matrices R and C **Dual Problem**					
	Παράδειγμα 1	**Παράδειγμα 2**	**Παράδειγμα 3**	**Παράδειγμα 4**	**Παράδειγμα 5**
Ισορροπία Nash	0.000541	0.001508	0.000234	0.000931	0.000244
Αρχική τιμή της f(x,y)	0.039285	0.239531	0.006022	0.088578	0.031969
Αριθμός Βημάτων	3	17	3	11	8

Σημείωση: Τα αντίστοιχα αρχεία εξόδου φαίνονται στα αρχεία κάτω από τον φάκελο \random\100x100_B

Παρατηρήσεις πάνω στα παίγνια για τυχαίους πίνακες κέρδους R και C:

- Αρχικά βλέπουμε ότι έχουμε πολύ καλές ισορροπίες Nash. Ακόμα κι αν δεν είναι όλες exact ισορροπίες Nash, είναι της τάξης του δ (10^{-3}) και μικρότερες, τόσο στην λύση του primal όσο και στην λύση του dual προβλήματος.
  Το αποτέλεσμα αυτό δεν έχει μεγάλη σχέση με το μέγεθος του προβλήματος, εφόσον παρατηρείται η ίδια συμπεριφορά σε όλα τα παίγνια, ανεξαρτήτου μεγέθους.
- Όταν στην primal λύση έχουμε αποτέλεσμα της τάξης του δ και μεγαλύτερης, τότε το dual δίνει καλύτερη λύση.
- Επίσης ο αριθμός των βημάτων μέχρι να τερματίσει ο αλγόριθμος είναι από 2 έως και 8 βήματα, ενώ για το dual πρόβλημα είναι συνήθως λιγότερα.
  Όταν χρησιμοποιήσαμε το δεύτερο κριτήριο τερματισμού τα βήματα είναι λίγο περισσότερα. Στα συγκεκριμένα παίγνια όμως δεν εμφανίζεται ποιοτική διαφορά στην ισορροπία μιας και έχουμε ήδη μια πολύ καλή συμπεριφορά.

Επιπλέον παρατηρήσεις στο τέλος του κεφαλαίου.

6.4 Πίνακες Κέρδους R και C για win-loose παίγνια (Matrices R and C for win-loose games) [Πρώτος τρόπος]

Στην προσπάθεια μας να δημιουργήσουμε δύσκολα παίγνια με τα χαρακτηριστικά που αναφέραμε παραπάνω, δημιουργήσαμε κάποιου είδους win-loose παιγνίων.

Χαρακτηριστικά παιγνίων:

- Οι πίνακες κέρδους R και C περιέχουν τιμές 0 ή 1 σε κάθε θέση τους.
  Με αυτόν τον τρόπο δημιουργούμε win-loose παίγνια.
  [*βλέπε ορισμό win-loose παιγνίων στο δεύτερο κεφάλαιο*]
- Αν σε κάποιο σημείο (i,j) του πίνακα R έχουμε R(i,j) = 1 τότε θέτουμε το C(i,j) = 0
  Με αυτόν τον τρόπο αποφεύγουμε την ύπαρξη αγνής ισορροπίας Nash, που θα ήταν εκεί όπου και οι δύο παίκτες θα είχαν κέρδος 1.
- Αντίστοιχα αν σε κάποιο σημείο (i,j) του πίνακα C έχουμε C(i,j)=1 τότε θέτουμε το R(i,j)=0
  Ομοίως, με αυτόν τον τρόπο αποφεύγουμε την ύπαρξη αγνής ισορροπίας Nash, που θα ήταν εκεί όπου και οι δύο παίκτες θα είχαν κέρδος 1.
- Ο συνολικός πίνακας R+C δεν έχει πολλούς "1", αλλά πολλά "0" σε τυχαίες θέσεις.

Οι πίνακες R και C για το συγκεκριμένο παίγνιο win-loose που περιγράψαμε παραπάνω, παράγονται μέσω κώδικα σε MatLab.

Ο κώδικας είναι ο παρακάτω:

```
%The size of matrices R and C [n x n]
n = …;

A = rand(n);
R = A > 0.7;

B = rand(n);
C = B > 0.7;

R(C)=0;
C(R)=0;

sum(sum(R+C == 1))

%Create the proper file that ANSI C code takes as input file
fid = fopen('C:\...\r_c_m.txt','w');
fprintf(fid,'%.16f \t',transpose(R));
fprintf(fid,'%.16f \t',transpose(C));
status = fclose(fid);
```

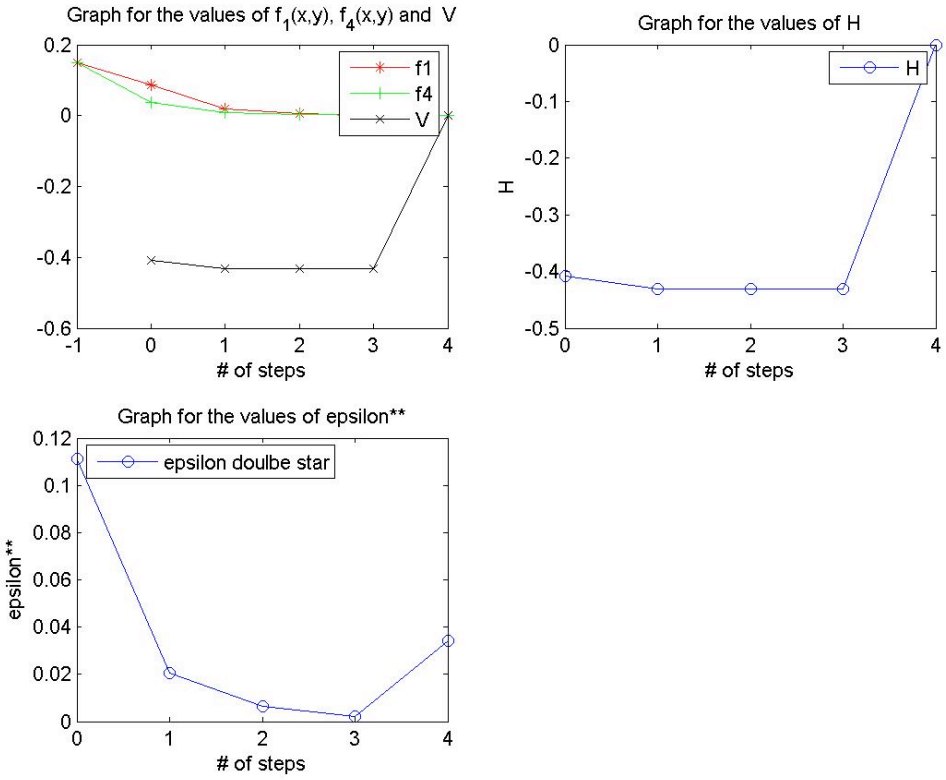
6.4.1 R και C πίνακες μεγέθους 10 x 10

Χαρακτηριστικά Παιγνίου:

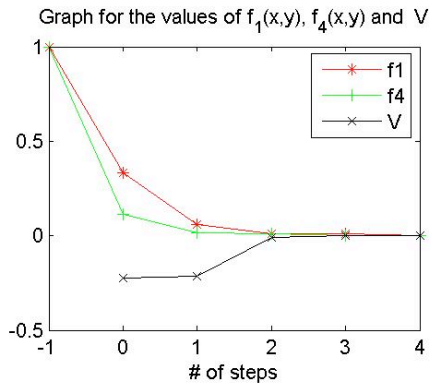
- Τύπος Παιγνίου: Win-loose παίγνιο
- Μέγεθος παιγνίου: 10 x 10
- Κριτήριο τερματισμού: $V(x,y)-f(x,y)\geq -\delta$
- Τιμή της παραμέτρου δ: 10^{-3}
- Αρχικές τιμές των x και y: Ομοιόμορφη κατανομή

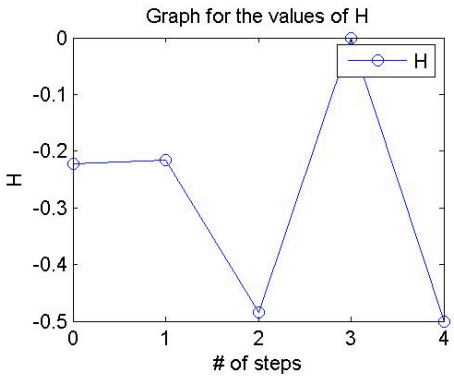
Παράδειγμα 1 [αρχείο: *\win-loose_first\10x10\1*]

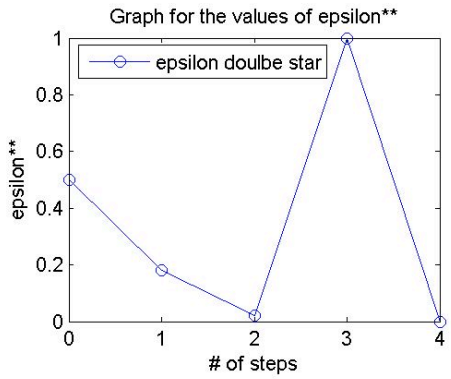
Primal Graph



Dual Graph







Πίνακες Αποτελεσμάτων Συνολικών Πειραμάτων

Matrices R and C for win-loose Game **Primal Problem**					
	Παράδειγμα 1	**Παράδειγμα 2**	**Παράδειγμα 3**	**Παράδειγμα 4**	**Παράδειγμα 5**
Ισορροπία Nash	0.000515	0	0	0	0
Αρχική τιμή της f(x,y)	0.150000	0.280000	0.300000	0.370000	0.270000
Αριθμός Βημάτων	5	1	2	2	8
Αρχικά x και y	Uniform	Uniform	Uniform	Uniform	Uniform

Matrices R and C for win-loose Game **Dual Problem**					
	Παράδειγμα 1	**Παράδειγμα 2**	**Παράδειγμα 3**	**Παράδειγμα 4**	**Παράδειγμα 5**
Ισορροπία Nash	0	0	0	0	0
Αρχική	1.000000	0	0.300000	0.045455	0.666667

τιμή της f(x,y)					
Αριθμός Βημάτων	5	1	2	2	5

Σημείωση: Οι αντίστοιχες γραφικές παραστάσεις και τα αρχεία εξόδου για τα υπόλοιπα παραδείγματα φαίνονται στα αρχεία κάτω από τον φάκελο \win-loose_first\10x10

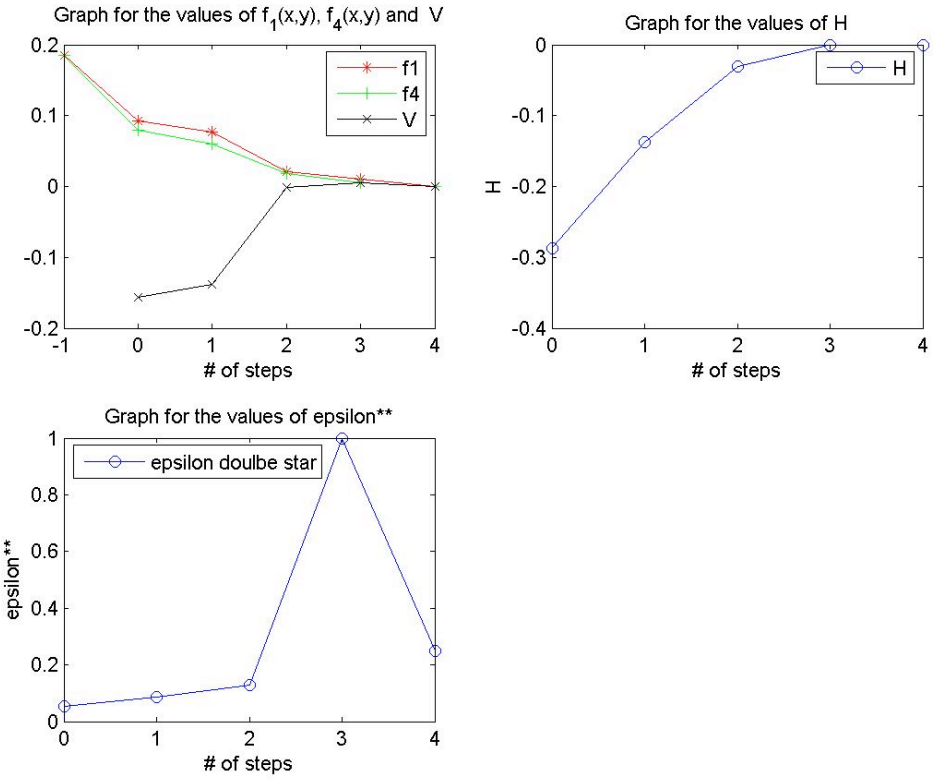
6.4.2 R και C πίνακες μεγέθους 20 x 20

Χαρακτηριστικά Παιγνίου:

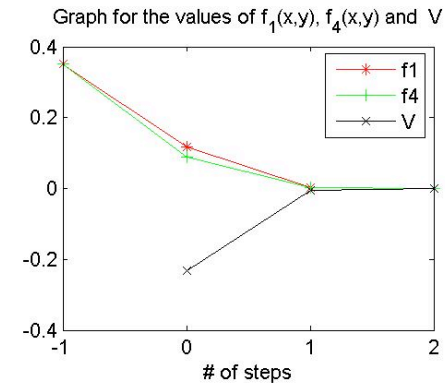
- Τύπος Παιγνίου: Win-loose παίγνιο
- Μέγεθος παιγνίου: 20 x 20
- Κριτήριο τερματισμού: $V(x,y)-f(x,y)\geq -\delta$
- Τιμή της παραμέτρου δ: 10^{-3}
- Αρχικές τιμές των x και y: Ομοιόμορφη κατανομή

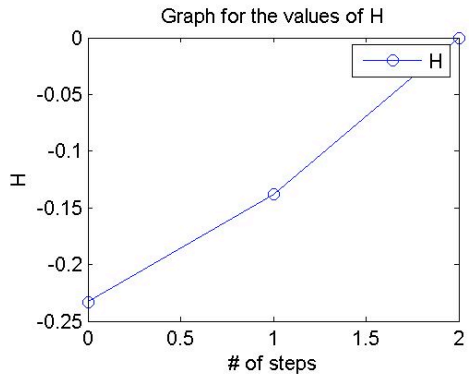
Παράδειγμα 1 [αρχείο: *\win-loose_first\20x20\1*]

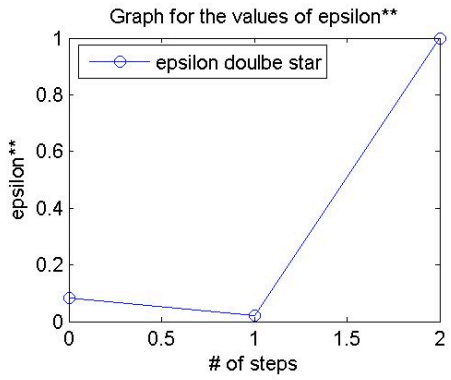
Primal Graph



Dual Graph







Πίνακες Αποτελεσμάτων Συνολικών Πειραμάτων

Matrices R and C for win-loose Game **Primal Problem**					
	Παράδειγμα 1	**Παράδειγμα 2**	**Παράδειγμα 3**	**Παράδειγμα 4**	**Παράδειγμα 5**
Ισορροπία Nash	0	0.003635	0.003245	0	0
Αρχική τιμή της f(x,y)	0.185000	0.317500	0.347500	0.242500	0.165000
Αριθμός Βημάτων	5	3	3	5	4
Αρχικά x και y	Uniform	Uniform	Uniform	Uniform	Uniform

Matrices R and C for win-loose Game **Dual Problem**					
	Παράδειγμα 1	**Παράδειγμα 2**	**Παράδειγμα 3**	**Παράδειγμα 4**	**Παράδειγμα 5**
Ισορροπία Nash	0	0	0	0.000627	0
Αρχική	0.350565	0.137855	0.12346	0.083333	0.181296

τιμή της f(x,y)					
Αριθμός Βημάτων	3	5	5	4	4

Σημείωση: Οι αντίστοιχες γραφικές παραστάσεις και τα αρχεία εξόδου για τα υπόλοιπα παραδείγματα φαίνονται στα αρχεία κάτω από τον φάκελο \win-loose_first\20x20

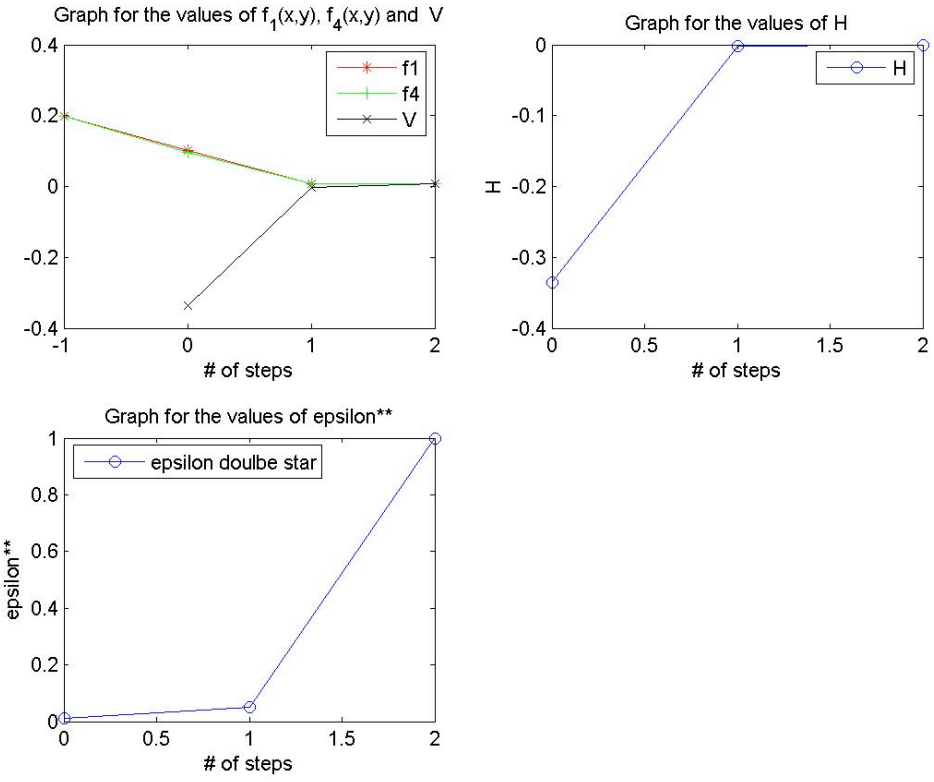
6.4.3 R και C πίνακες μεγέθους 30 x 30

Χαρακτηριστικά Παιγνίου:

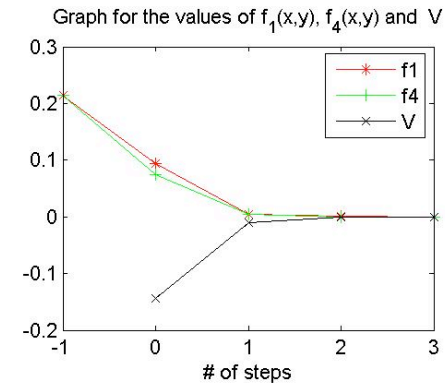
- Τύπος Παιγνίου: Win-loose παίγνιο
- Μέγεθος παιγνίου: 30 x 30
- Κριτήριο τερματισμού: $V(x,y)-f(x,y)\geq -\delta$
- Τιμή της παραμέτρου δ: 10^{-3}
- Αρχικές τιμές των x και y: Ομοιόμορφη κατανομή

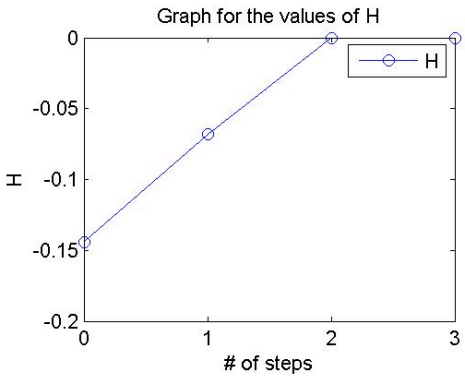
Παράδειγμα 1 [αρχείο: *\win-loose_first\30x30\1*]

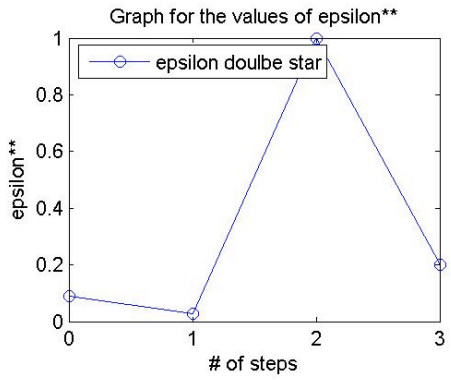
Primal Graph



Dual Graph







Πίνακες Αποτελεσμάτων Συνολικών Πειραμάτων

Matrices R and C for win-loose Game **Primal Problem**					
	Παράδειγμα 1	**Παράδειγμα 2**	**Παράδειγμα 3**	**Παράδειγμα 4**	**Παράδειγμα 5**
Ισορροπία Nash	0.007317	0	0	0	0
Αρχική τιμή της f(x,y)	0.198889	0.284444	0.210000	0.146667	0.241111
Αριθμός Βημάτων	3	3	6	5	4
Αρχικά x και y	Uniform	Uniform	Uniform	Uniform	Uniform

Matrices R and C for win-loose Game **Dual Problem**					
	Παράδειγμα 1	**Παράδειγμα 2**	**Παράδειγμα 3**	**Παράδειγμα 4**	**Παράδειγμα 5**
Ισορροπία Nash	0	0	0	0	0
Αρχική	0.213773	0.014926	0.146909	0.175391	0.206061

τιμή της f(x,y)					
Αριθμός Βημάτων	4	2	2	7	4

Σημείωση: Οι αντίστοιχες γραφικές παραστάσεις και τα αρχεία εξόδου για τα υπόλοιπα παραδείγματα φαίνονται στα αρχεία κάτω από τον φάκελο \win-loose_first\30x30

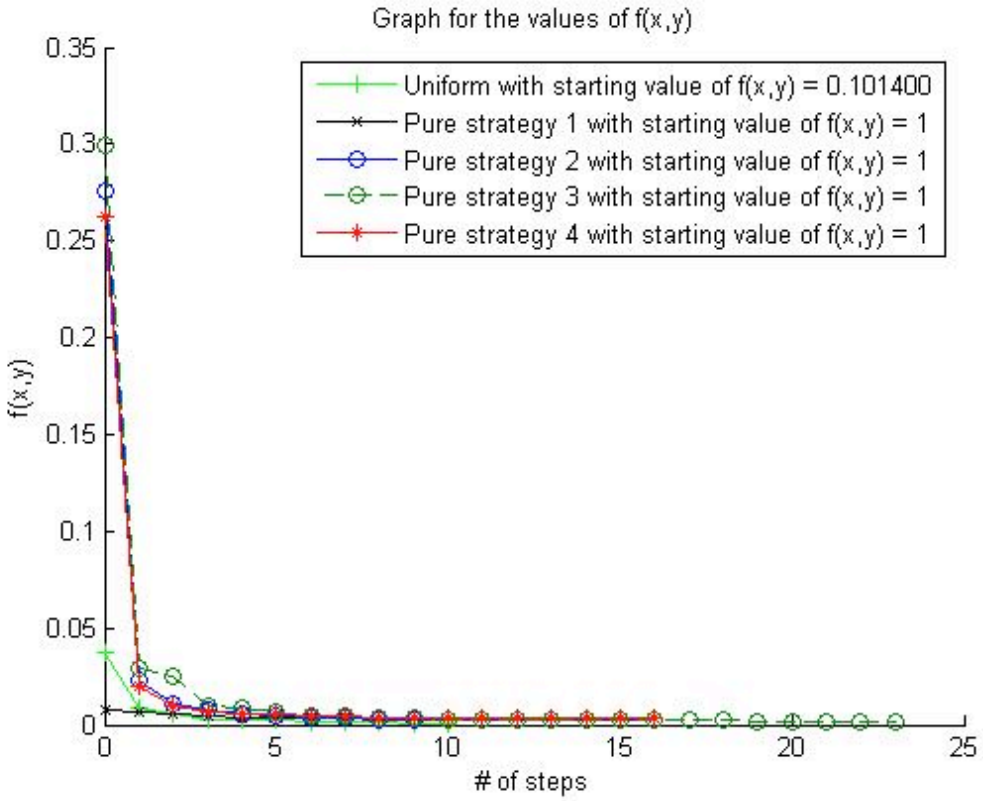
6.4.4 R και C πίνακες μεγέθους 100 x 100

Χαρακτηριστικά Παιγνίου:

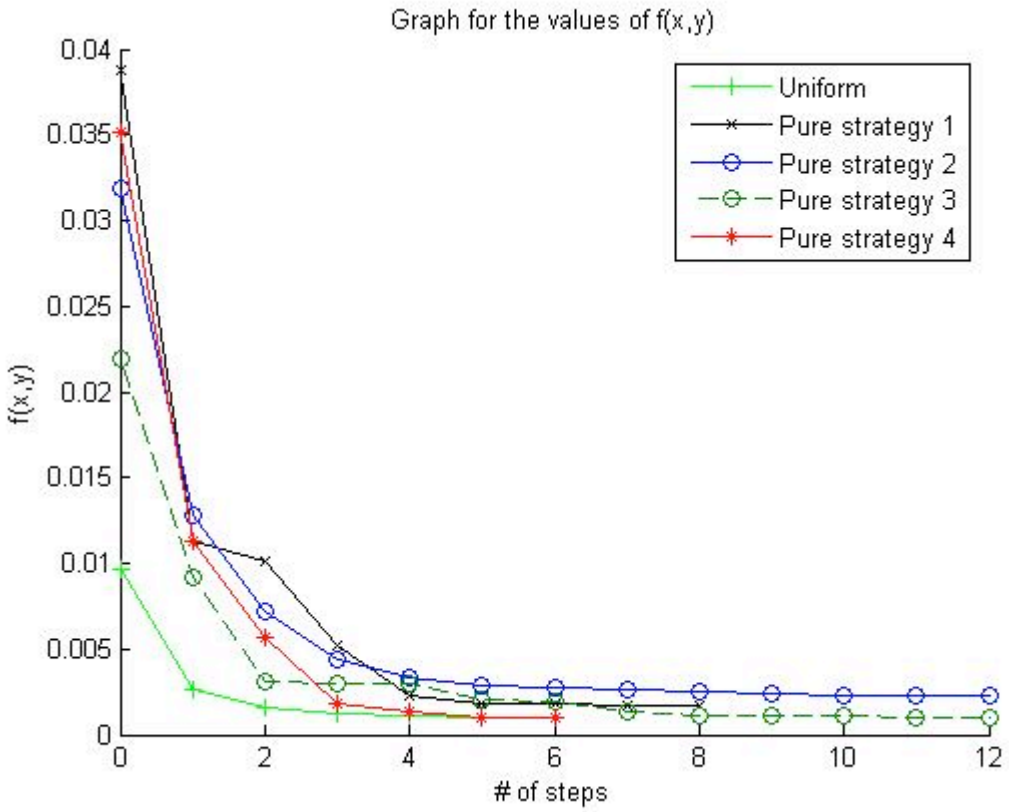
- Τύπος Παιγνίου: Win-loose παίγνιο
- Μέγεθος παιγνίου: 100 x 100
- Κριτήριο τερματισμού: $f_4 \succ f_1(1-(\frac{\delta}{1+\delta})^2)$
- Τιμή της παραμέτρου δ: 10^{-3}
- Αρχικές τιμές των x και y: Ομοιόμορφη κατανομή και αγνές στρατηγικές

Παράδειγμα 1 [αρχείο: *\win-loose_first\100x100*]

Primal Graph



Dual Graph



Πίνακες Αποτελεσμάτων Πειράματος

Matrices R and C for win-loose Game **Primal Problem**					
	Παράδειγμα 1	**Παράδειγμα 2**	**Παράδειγμα 3**	**Παράδειγμα 4**	**Παράδειγμα 5**
Ισορροπία Nash	0.000766	0.003218	0.002159	0.001795	0.003386
Αρχική τιμή της f(x,y)	0.101400	1.000000	1.000000	1.000000	1.000000
Αριθμός Βημάτων	11	9	17	24	17
Αρχικά x και y	Uniform	Αγνή Στρατηγική	Αγνή Στρατηγική	Αγνή Στρατηγική	Αγνή Στρατηγική

Matrices R and C for win-loose Game **Dual Problem**					
	Παράδειγμα 1	**Παράδειγμα 2**	**Παράδειγμα 3**	**Παράδειγμα 4**	**Παράδειγμα 5**
Ισορροπία Nash	0.000960	0.001642	0.002233	0.000966	0.00098

Αρχική τιμή της f(x,y)	0.022445	0.125594	0.050702	0.035484	0.080064
Αριθμός Βημάτων	6	9	13	13	7

Σημείωση: Οι αντίστοιχες γραφικές παραστάσεις και τα αρχεία εξόδου για τα υπόλοιπα παραδείγματα φαίνονται στα αρχεία κάτω από τον φάκελο \win-loose_first\100x100

Παρατηρήσεις πάνω στα win-loose παίγνια:

- Αρχικά βλέπουμε ότι κι εδώ έχουμε πολύ καλές ισορροπίες Nash.
  Αυτό σημαίνει ότι δεν μπορέσαμε να δημιουργήσαμε πολύ δύσκολα παίγνια για τον αλγόριθμό μας με αυτόν τον τρόπο.
  Παρ` όλα αυτά παρατηρούμε ότι καθώς αυξάνεται το μέγεθος των παιγνίων το πρόβλημα αρχίζει να δυσκολεύει ελάχιστα. Με την έννοια ότι στα μικρά παίγνια παρατηρήθηκε μεγάλη εμφάνιση exact ισορροπίας Nash, ενώ καθώς το πρόβλημα γινόταν πιο μεγάλο εμφανίζονταν και ισορροπίες Nash της τάξης του δ (10^{-3}) και μικρότερες, τόσο στην λύση του primal όσο και στην λύση του dual προβλήματος.
- Όταν στην primal λύση έχουμε αποτέλεσμα της τάξης του δ και μεγαλύτερης, τότε το dual δίνει καλύτερη λύση.
- Επίσης ο αριθμός των βημάτων μέχρι να τερματίσει ο αλγόριθμος είναι από 2 έως και 8 βήματα (κυρίως κοντά στα 4 μόνο βήματα), ενώ για το dual πρόβλημα είναι συνήθως λιγότερα.
  Όταν χρησιμοποιήσαμε το δεύτερο κριτήριο τερματισμού τα βήματα είναι σαφώς περισσότερα (10-25 βήματα). Στα συγκεκριμένα παίγνια όμως δεν εμφανίζεται ποιοτική διαφορά στην ισορροπία μιας και έχουμε ήδη μια πολύ καλή συμπεριφορά. Παρ` όλα αυτά παρατηρήσαμε ότι τα περισσότερα από αυτά τα βήματα δεν αλλάζουν ουσιαστικά την τιμή της συνάρτησης f(x,y). Πράγμα που σημαίνει ότι με καλύτερη επιλογή του ε, ενδεχομένως αυτά τα βήματα να μην υπήρχαν.

Επιπλέον παρατηρήσεις στο τέλος του κεφαλαίου

6.5 Πίνακες Κέρδους R και C για win-loose παίγνια (Matrices R and C for win-loose games) [Δεύτερος τρόπος]

Ένας άλλος τρόπος δημιουργίας δύσκολων win-loose παιγνίων με τα χαρακτηριστικά που αναφέραμε παραπάνω, περιγράφεται παρακάτω.

Χαρακτηριστικά παιγνίων:
Οι πίνακες R και C έχουν τα εξής χαρακτηριστικά:

- Ο πίνακας C είναι ο I, όπου I είναι ο nxn identity matrix, δηλαδή C(i,i)=1 για κάθε i και όλα τα υπόλοιπα 0.
- Ο πίνακας R είναι ο ένα πίνακας μεγέθους nxn και περιέχει μόνο τιμές 0 ή 1.
- Ο πίνακας R έχει τα διαγώνια στοιχεία του ίσα 0 και αν για ένα μη διαγώνιο στοιχείο ισχύει R(i,j)=1, τότε το R(j,i)=0, δηλαδή να έχουμε πάντα R(i,j)+ R(j,i) = 0 ή 1 το πολύ.

  Αυτό το καταφέρνουμε διατρέχοντας όλα τις θέσεις των στοιχείων που υπάρχουν πάνω από την διαγώνιο και αποφασίζοντας με πιθανότητα 50% αν θα μπει "1" ή "0".
  Έστω ότι είμαστε στη θέση (i,j). Αν μπει 0 τότε προχωράμε στην επόμενη θέση του πίνακα R.
  Αν επιλεγεί το 1, τότε με πιθανότητα 50% επιλέγουμε αν αυτό θα τοποθετεί στην θέση (i,j) ή στην (j,i).
- Το πλήθος των "1" στον R είναι περίπου μεταξύ του n(n-1)/4 και του n(n-1)/3.

Οι πίνακες R και C για το συγκεκριμένο παίγνιο win-loose που περιγράψαμε παραπάνω, παράγονται μέσω κώδικα σε MatLab.

Ο κώδικας είναι ο παρακάτω:

```
%The size of matrices R and C [n x n]
n = …

C = eye(n);
A = rand(n);
B = rand(n);
R = zeros(n);

for i=1:n
   for j=i+1:n
      if A(i,j) > 0.5
         if B(i,j) > 0.5
            R(i,j) = 1.0;
         else
            R(j,i) = 1.0;
         end
      end
   end
end
```

```
sum(sum(R == 1))

%Create the proper file that ANSI C code takes as input file
fid = fopen('C:\...\r_c_m.txt','w');
fprintf(fid,'%.16f \t',transpose(R));
fprintf(fid,'%.16f \t',transpose(C));
status = fclose(fid);
```

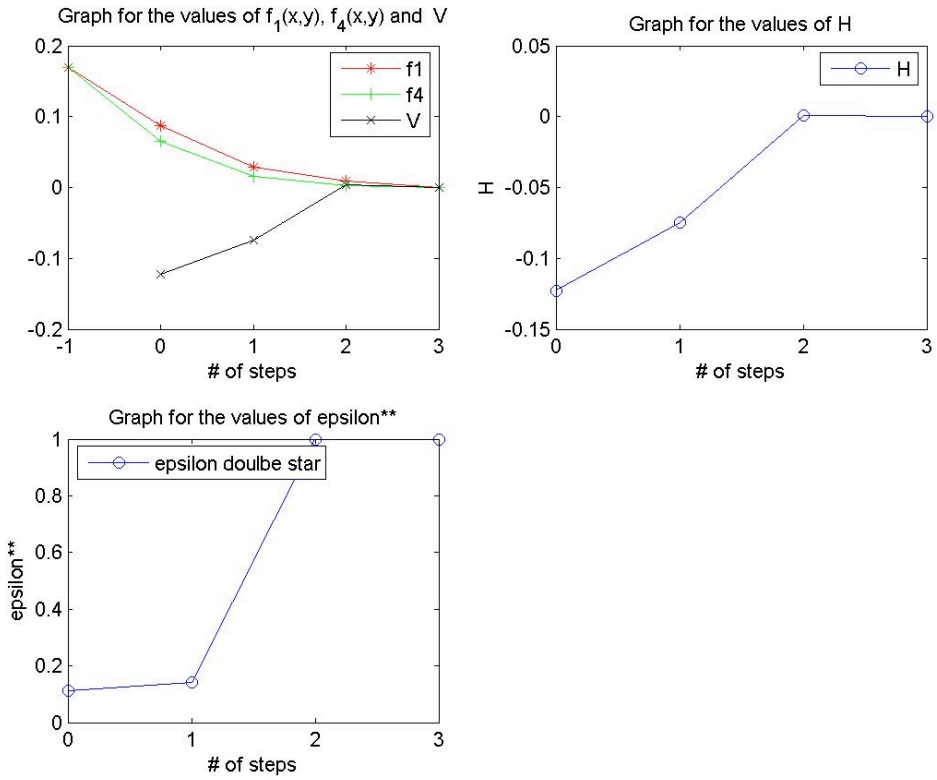
6.5.1 R και C πίνακες μεγέθους 10 x 10

Χαρακτηριστικά Παιγνίου:

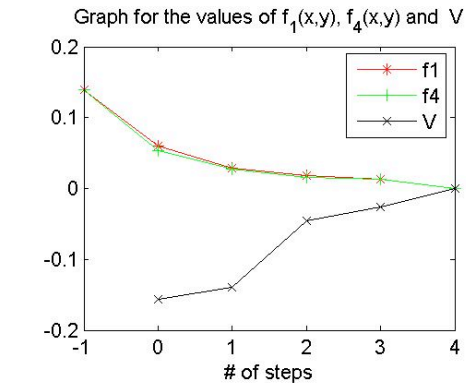
- Τύπος Παιγνίου: Win-loose παίγνιο
- Μέγεθος παιγνίου: 10 x 10
- Κριτήριο τερματισμού: $V(x,y)-f(x,y) \geq -\delta$
- Τιμή της παραμέτρου δ: 10^{-3}
- Αρχικές τιμές των x και y: Ομοιόμορφη κατανομή

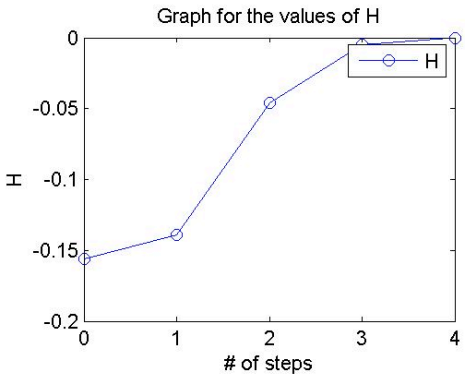
Παράδειγμα 1 [αρχείο: *win-loose_second\10\1*]

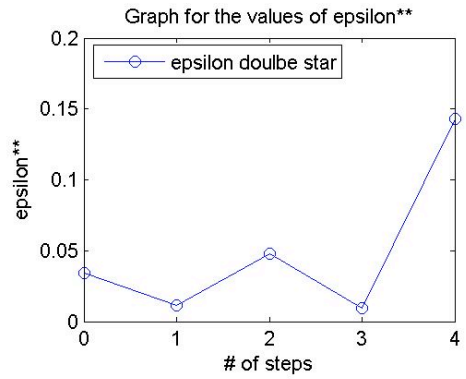
Primal Graph



Dual Graph







Πίνακες Αποτελεσμάτων Συνολικών Πειραμάτων

Matrices R and C for win-loose Game **Primal Problem**					
	Παράδειγμα 1	**Παράδειγμα 2**	**Παράδειγμα 3**	**Παράδειγμα 4**	**Παράδειγμα 5**
Ισορροπία Nash	0	0	0	0	0
Αρχική τιμή της f(x,y)	0.170000	0.130000	0.280000	0.390000	0.200000
Αριθμός Βημάτων	4	4	2	7	4
Αρχικά x και y	Uniform	Uniform	Uniform	Uniform	Uniform

Matrices R and C for win-loose Game **Dual Problem**					
	Παράδειγμα 1	**Παράδειγμα 2**	**Παράδειγμα 3**	**Παράδειγμα 4**	**Παράδειγμα 5**
Ισορροπία Nash	0	0.000254	0	0	0
Αρχική	0.138889	0.111330	0.111111	0.600000	0.250000

τιμή της f(x,y)					
Αριθμός Βημάτων	5	5	3	5	8

Σημείωση: Οι αντίστοιχες γραφικές παραστάσεις και τα αρχεία εξόδου για τα υπόλοιπα παραδείγματα φαίνονται στα αρχεία κάτω από τον φάκελο / win-loose_second

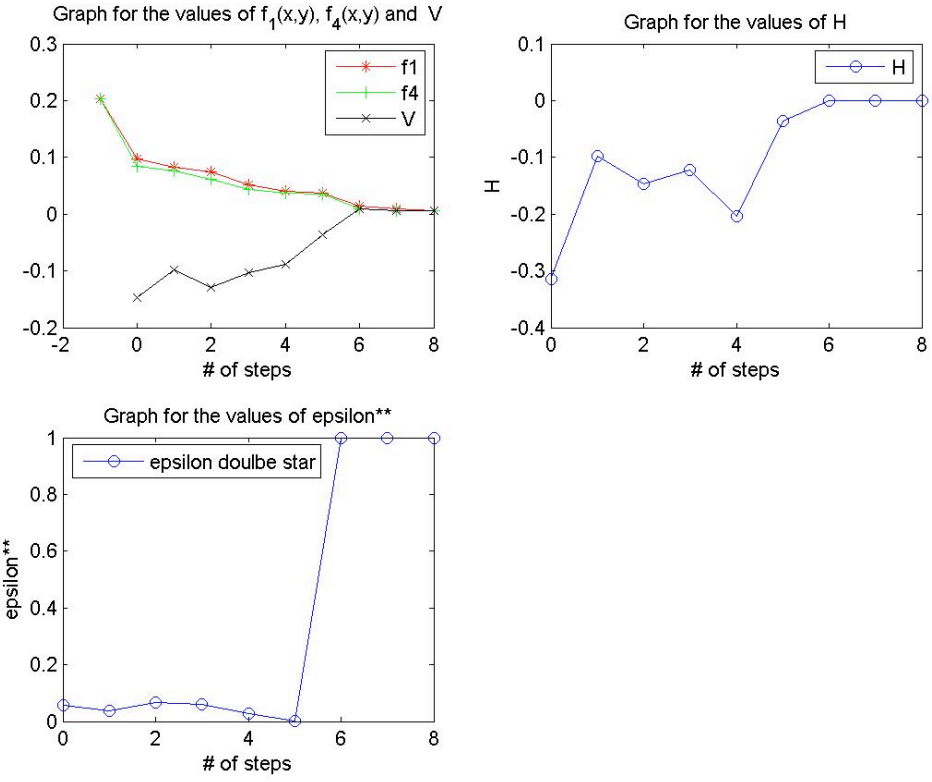
6.5.2 R και C πίνακες μεγέθους 20 x 20

Χαρακτηριστικά Παιγνίου:

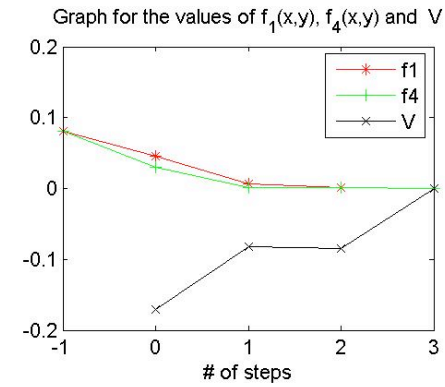
- Τύπος Παιγνίου: Win-loose παίγνιο
- Μέγεθος παιγνίου: 20 x 20
- Κριτήριο τερματισμού: $V(x,y)-f(x,y)\geq -\delta$
- Τιμή της παραμέτρου δ: 10^{-3}
- Αρχικές τιμές των x και y: Ομοιόμορφη κατανομή

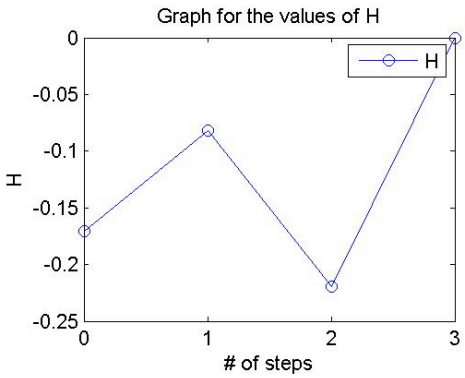
Παράδειγμα 1 [αρχείο: *win-loose_second \20\1*]

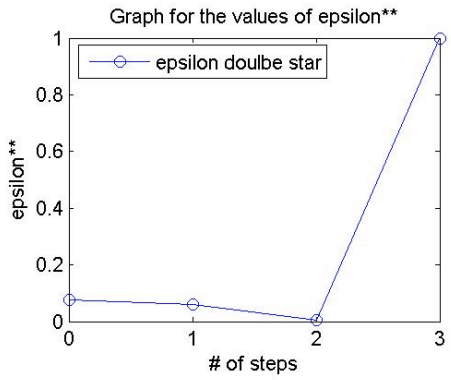
Primal Graph



Dual Graph







Πίνακες Αποτελεσμάτων Συνολικών Πειραμάτων

Matrices R and C for win-loose Game **Primal Problem**					
	Παράδειγμα 1	**Παράδειγμα 2**	**Παράδειγμα 3**	**Παράδειγμα 4**	**Παράδειγμα 5**
Ισορροπία Nash	0.005847	0.007261	0	0	0.002481
Αρχική τιμή της f(x,y)	0.202500	0.100000	0.167500	0.230000	0.117500
Αριθμός Βημάτων	9	4	2	5	6
Αρχικά x και y	Uniform	Uniform	Uniform	Uniform	Uniform

Matrices R and C for win-loose Game **Dual Problem**					
	Παράδειγμα 1	**Παράδειγμα 2**	**Παράδειγμα 3**	**Παράδειγμα 4**	**Παράδειγμα 5**
Ισορροπία Nash	0	0	0	0.003793	0

Αρχική τιμή της f(x,y)	0.080542	0.082147	0.250293	0.151697	0.091256
Αριθμός Βημάτων	4	2	3	6	3

Σημείωση: Οι αντίστοιχες γραφικές παραστάσεις και τα αρχεία εξόδου για τα υπόλοιπα παραδείγματα φαίνονται στα αρχεία κάτω από τον φάκελο / win-loose_second

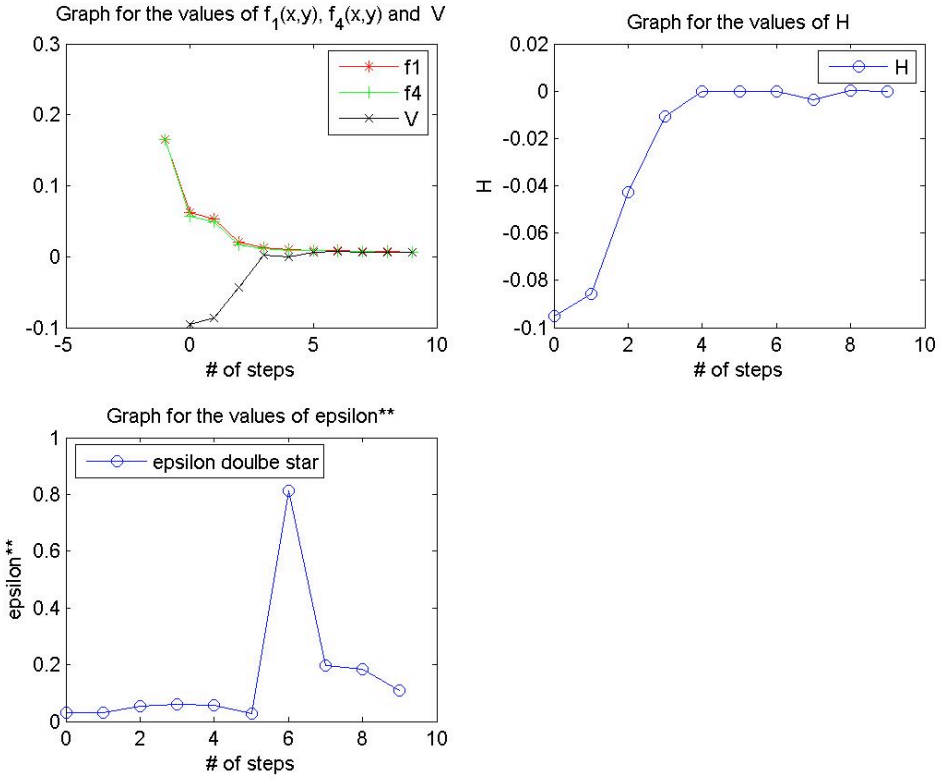
6.5.3 R και C πίνακες μεγέθους 30 x 30

Χαρακτηριστικά Παιγνίου:

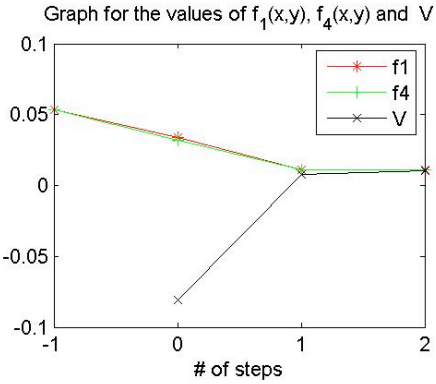
- Τύπος Παιγνίου: Win-loose παίγνιο
- Μέγεθος παιγνίου: 30 x 30
- Κριτήριο τερματισμού: $V(x,y)-f(x,y)\geq -\delta$
- Τιμή της παραμέτρου δ: 10^{-3}
- Αρχικές τιμές των x και y: Ομοιόμορφη κατανομή

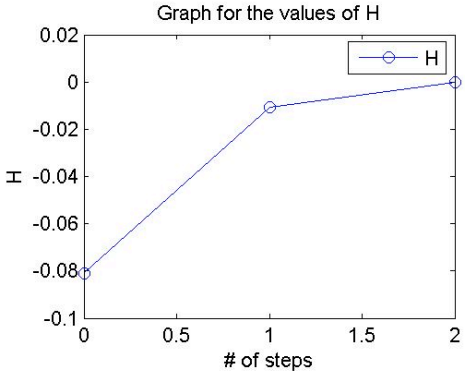
Παράδειγμα 1 [αρχείο: *win-loose_second \30\1*]

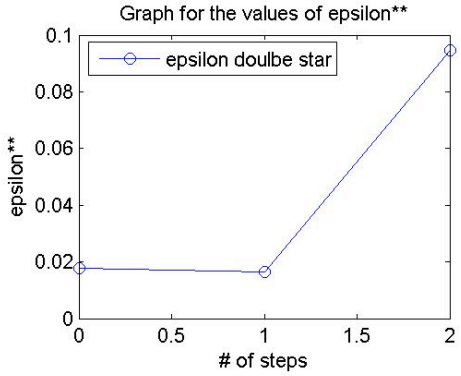
Primal Graph



Dual Graph







Πίνακες Αποτελεσμάτων Συνολικών Πειραμάτων

Matrices R and C for win-loose Game **Primal Problem**					
	Παράδειγμα 1	**Παράδειγμα 2**	**Παράδειγμα 3**	**Παράδειγμα 4**	**Παράδειγμα 5**
Ισορροπία Nash	0.006803	0.005115	0.003566	0.003628	0.006237
Αρχική τιμή της f(x,y)	0.165556	0.195556	0.177778	0.118889	0.157778
Αριθμός Βημάτων	10	9	7	8	6
Αρχικά x και y	Uniform	Uniform	Uniform	Uniform	Uniform

Matrices R and C for win-loose Game **Dual Problem**					
	Παράδειγμα 1	**Παράδειγμα 2**	**Παράδειγμα 3**	**Παράδειγμα 4**	**Παράδειγμα 5**
Ισορροπία Nash	0.011096	0.00505	0.003036	0.001516	0.000313

Αρχική τιμή της f(x,y)	0.053672	0.060717	0.073299	0.108119	0.067634
Αριθμός Βημάτων	3	7	6	7	4

Σημείωση: Οι αντίστοιχες γραφικές παραστάσεις και τα αρχεία εξόδου για τα υπόλοιπα παραδείγματα φαίνονται στα αρχεία κάτω από τον φάκελο / win-loose_second

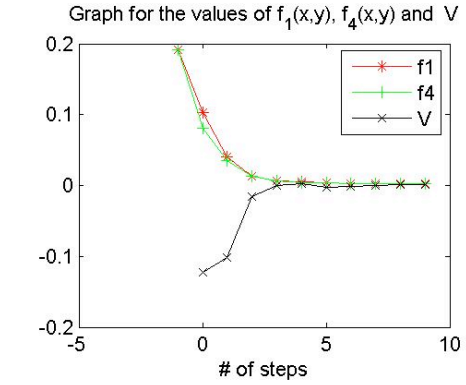
6.5.4 R και C πίνακες μεγέθους 50 x 50

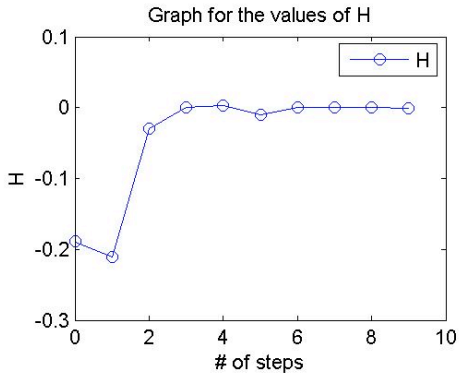
Χαρακτηριστικά Παιγνίου:

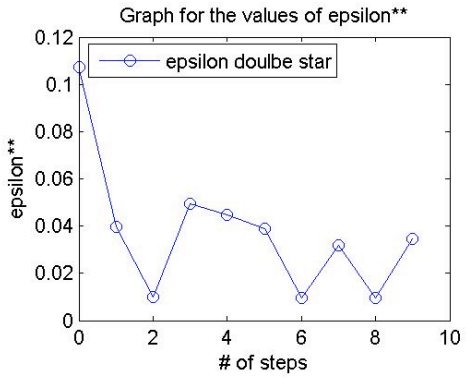
- Τύπος Παιγνίου: Win-loose παίγνιο
- Μέγεθος παιγνίου: 50 x 50
- Κριτήριο τερματισμού: $V(x,y)-f(x,y)\geq -\delta$
- Τιμή της παραμέτρου δ: 10^{-3}
- Αρχικές τιμές των x και y: Ομοιόμορφη κατανομή

Παράδειγμα 1 [αρχείο: *win-loose_second \50\1*]

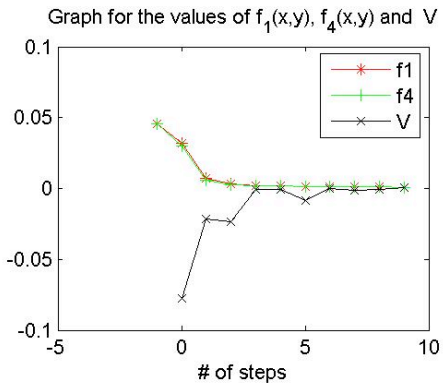
Primal Graph

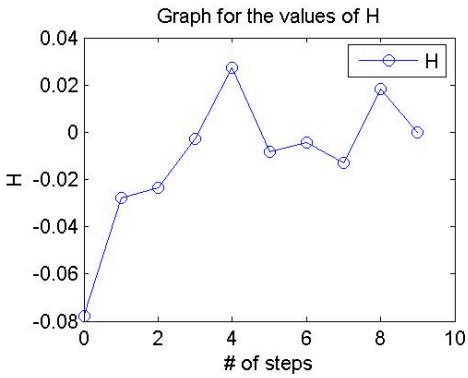


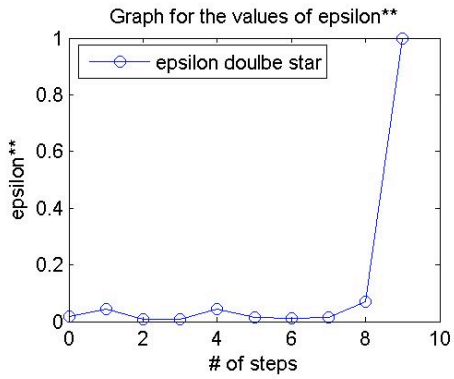




Dual Graph







Πίνακες Αποτελεσμάτων Συνολικών Πειραμάτων

Matrices R and C for win-loose Game **Primal Problem**					
	Παράδειγμα 1	**Παράδειγμα 2**	**Παράδειγμα 3**		
Ισορροπία Nash	0.002279	0.003174	0.001565		
Αρχική τιμή της f(x,y)	0.191600	0.146400	0.119200		
Αριθμός Βημάτων	10	9	7		
Αρχικά x και y	Uniform	Uniform	Uniform		

Matrices R and C for win-loose Game **Dual Problem**					
	Παράδειγμα 1	**Παράδειγμα 2**	**Παράδειγμα 3**		
Ισορροπία Nash	0.000766	0.004093	0.001126		

Αρχική τιμή της f(x,y)	0.045826	0.069342	0.051839		
Αριθμός Βημάτων	10	9	6		

Σημείωση: Οι αντίστοιχες γραφικές παραστάσεις και τα αρχεία εξόδου για τα υπόλοιπα παραδείγματα φαίνονται στα αρχεία κάτω από τον φάκελο / win-loose_second

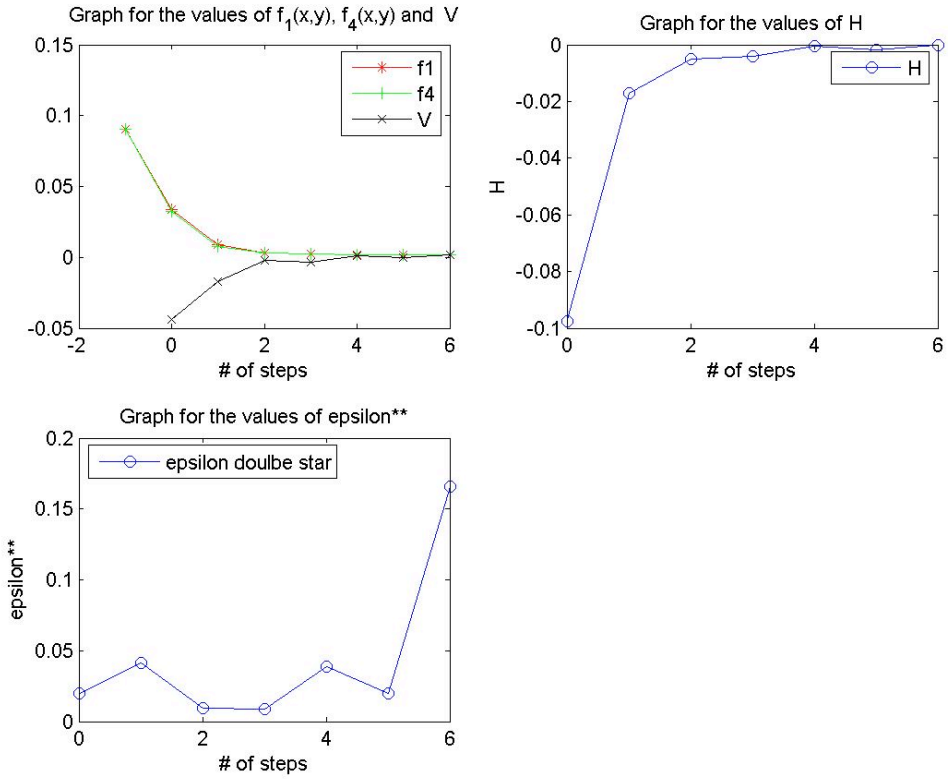
6.5.5 R και C πίνακες μεγέθους 100 x 100

Χαρακτηριστικά Παιγνίου:

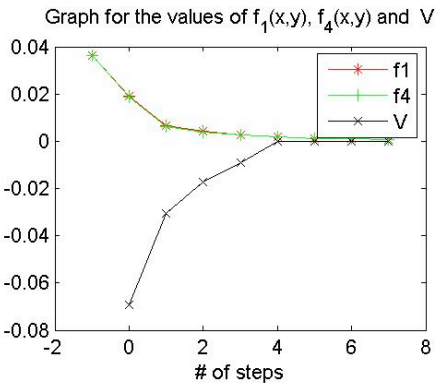
- Τύπος Παιγνίου: Win-loose παίγνιο
- Μέγεθος παιγνίου: 100 x 100
- Κριτήριο τερματισμού: $V(x,y)-f(x,y)\geq -\delta$
- Τιμή της παραμέτρου δ: 10^{-3}
- Αρχικές τιμές των x και y: Ομοιόμορφη κατανομή

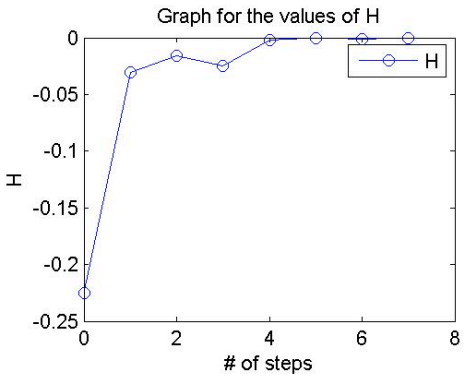
Παράδειγμα 1 [αρχείο: *win-loose_second \100\1*]

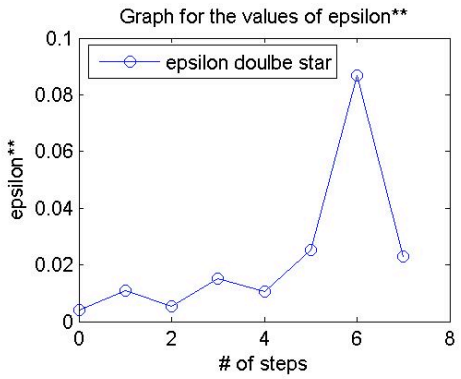
Primal Graph



Dual Graph







Πίνακες Αποτελεσμάτων Συνολικών Πειραμάτων

Matrices R and C for win-loose Game **Primal Problem**					
	Παράδειγμα 1	**Παράδειγμα 2**			
Ισορροπία Nash	0.001728	0.002023			
Αρχική τιμή της f(x,y)	0.090700	0.107800			
Αριθμός Βημάτων	7	15			
Αρχικά x και y	Uniform	Uniform			

Matrices R and C for win-loose Game **Dual Problem**					
	Παράδειγμα 1	**Παράδειγμα 2**			
Ισορροπία Nash	0.000665	0.001366			
Αρχική	0.036438	0.041922			

τιμή της f(x,y)					
Αριθμός Βημάτων	8	8			

Σημείωση: Οι αντίστοιχες γραφικές παραστάσεις και τα αρχεία εξόδου για τα υπόλοιπα παραδείγματα φαίνονται στα αρχεία κάτω από τον φάκελο / win-loose_second

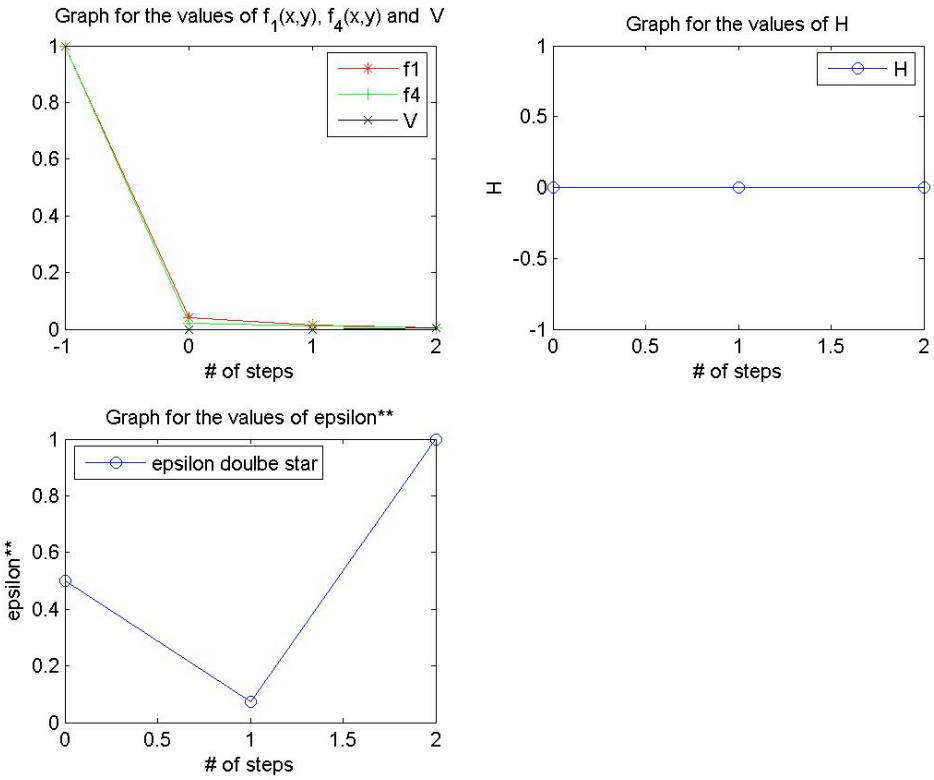
6.5.6 R και C πίνακες μεγέθους 50 x 50

Χαρακτηριστικά Παιγνίου:

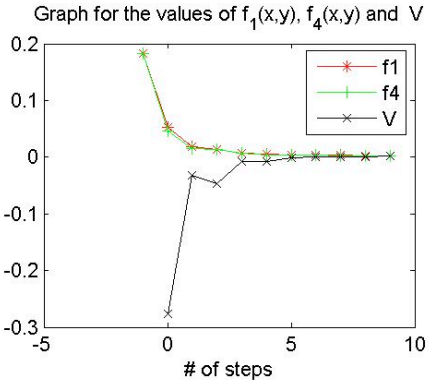
- Τύπος Παιγνίου: Win-loose παίγνιο
- Μέγεθος παιγνίου: 50 x 50
- Κριτήριο τερματισμού: $V(x,y)-f(x,y)\geq -\delta$
- Τιμή της παραμέτρου δ: 10^{-3}
- Αρχικές τιμές των x και y: Αγνές στρατηγικές: x = y = [1 0 … 0]

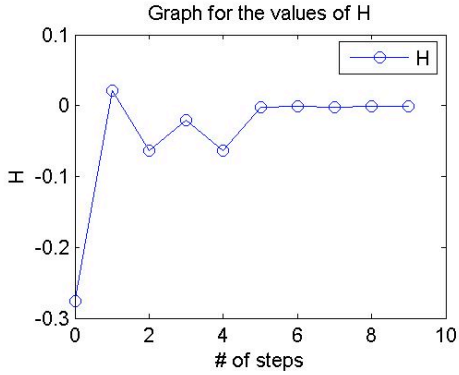
Παράδειγμα 1 [αρχείο: *win-loose_second \50\1b*]

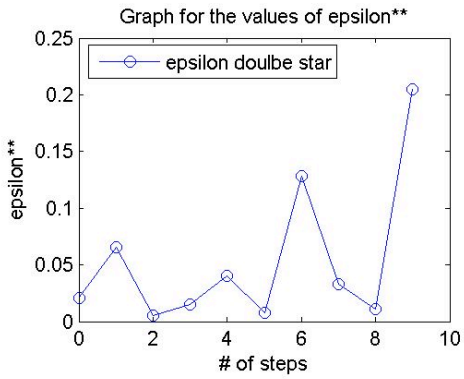
Primal Graph



Dual Graph







Πίνακες Αποτελεσμάτων Συνολικών Πειραμάτων

Matrices R and C for win-loose Game **Primal Problem**					
	Παράδειγμα 1	**Παράδειγμα 2**	**Παράδειγμα 3**		
Ισορροπία Nash	0.005291	0.018534	0.00204		
Αρχική τιμή της f(x,y)	1.000000	1.000000	1.000000		
Αριθμός Βημάτων	3	3	11		
Αρχικά x και y					

Matrices R and C for win-loose Game **Dual Problem**					
	Παράδειγμα 1	**Παράδειγμα 2**	**Παράδειγμα 3**		
Ισορροπία Nash	0.002092	0.004075	0.0017		

Αρχική τιμή της f(x,y)	0.183031	0.194542	0.047358		
Αριθμός Βημάτων	10	6	7		

Σημείωση: Οι αντίστοιχες γραφικές παραστάσεις και τα αρχεία εξόδου για τα υπόλοιπα παραδείγματα φαίνονται στα αρχεία κάτω από τον φάκελο / win-loose_second

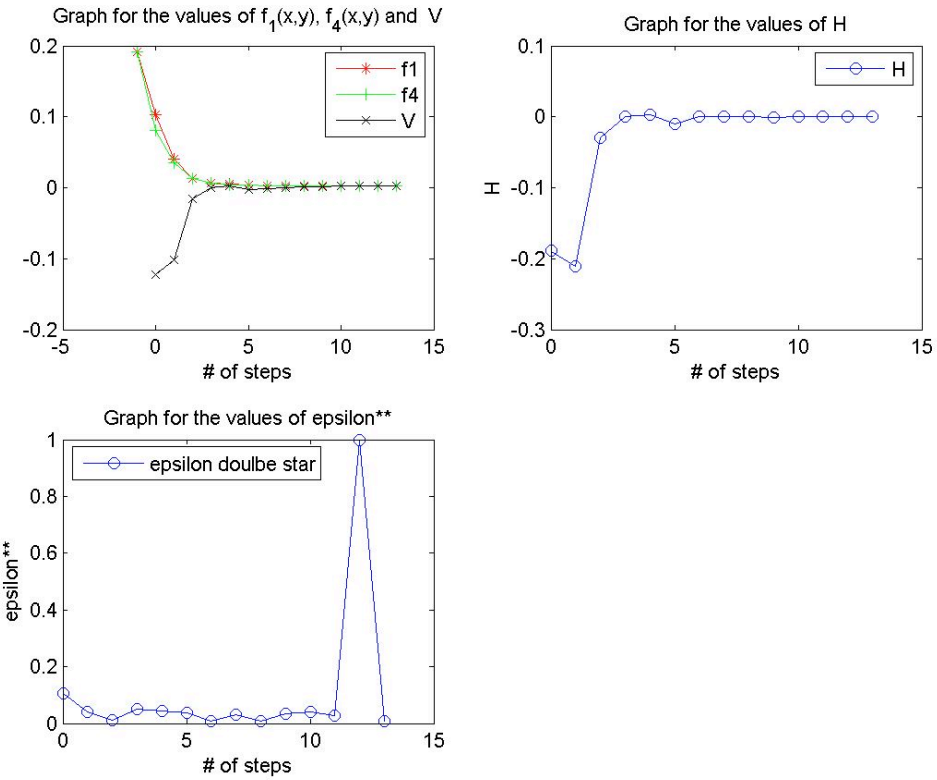
6.5.7 R και C πίνακες μεγέθους 50 x 50

Χαρακτηριστικά Παιγνίου:

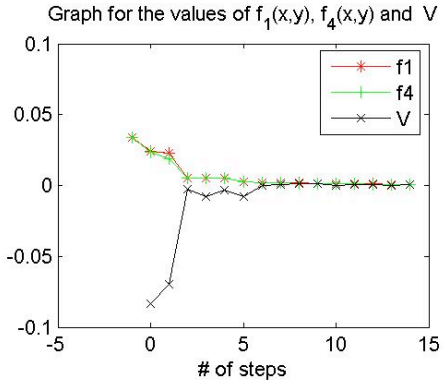
- Τύπος Παιγνίου: Win-loose παίγνιο
- Μέγεθος παιγνίου: 50 x 50
- Κριτήριο τερματισμού: $f4 > f1 * [1-(\delta/(1+\delta))^2$
- Τιμή της παραμέτρου δ: 10^{-3}
- Αρχικές τιμές των x και y: Ομοιόμορφη Κατανομή

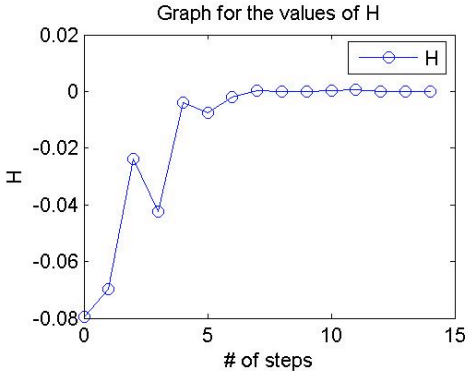
Παράδειγμα 1 [αρχείο: *win-loose_second \50\1c*]

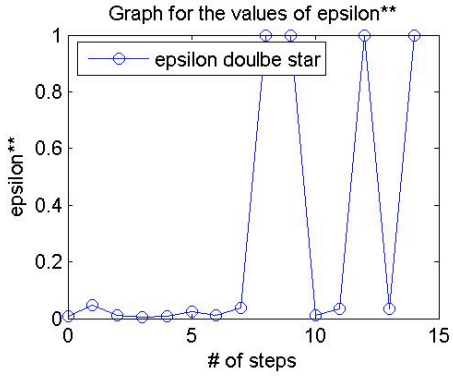
Primal Graph



Dual Graph







Πίνακες Αποτελεσμάτων Συνολικών Πειραμάτων

Matrices R and C for win-loose Game **Primal Problem**					
	Παράδειγμα 1	**Παράδειγμα 2**	**Παράδειγμα 3**		
Ισορροπία Nash	0.002224	0.002829	0.000671		
Αρχική τιμή της f(x,y)	0.191600	0.146400	0.119200		
Αριθμός Βημάτων	14	11	12		
Αρχικά x και y	Uniform	Uniform	Uniform		

Matrices R and C for win-loose Game **Dual Problem**					
	Παράδειγμα 1	**Παράδειγμα 2**	**Παράδειγμα 3**		
Ισορροπία	0.000692	0.001602	0.000745		

Nash					
Αρχική τιμή της f(x,y)	0.033931	0.072723	0.028419		
Αριθμός Βημάτων	15	15	6		

Σημείωση: Οι αντίστοιχες γραφικές παραστάσεις και τα αρχεία εξόδου για τα υπόλοιπα παραδείγματα φαίνονται στα αρχεία κάτω από τον φάκελο / win-loose_second

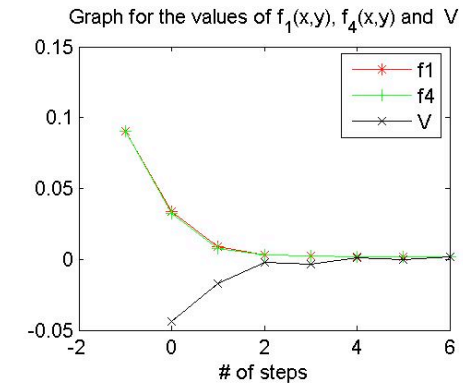
6.5.8 R και C πίνακες μεγέθους 100 x 100

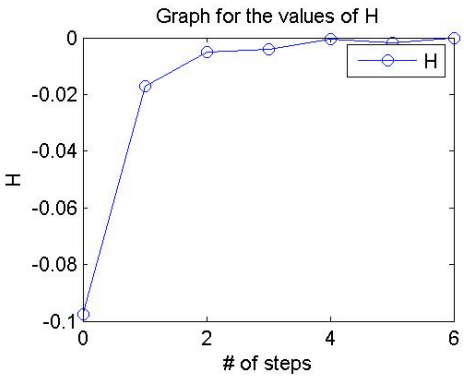
Χαρακτηριστικά Παιγνίου:

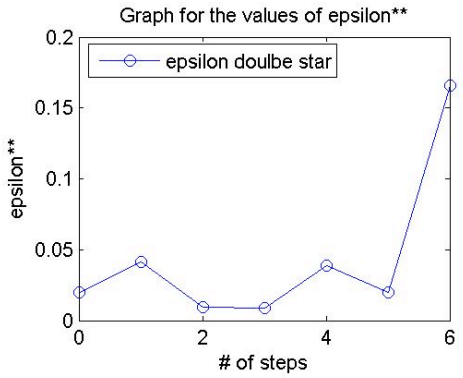
- Τύπος Παιγνίου: Win-loose παίγνιο
- Μέγεθος παιγνίου: 100 x 100
- Κριτήριο τερματισμού: $V(x,y)-f(x,y)\geq -\delta$
- Τιμή της παραμέτρου δ: 10^{-3}
- Αρχικές τιμές των x και y: Ομοιόμορφη Κατανομή

Παράδειγμα 1 [αρχείο: *win-loose_second \100\1*]

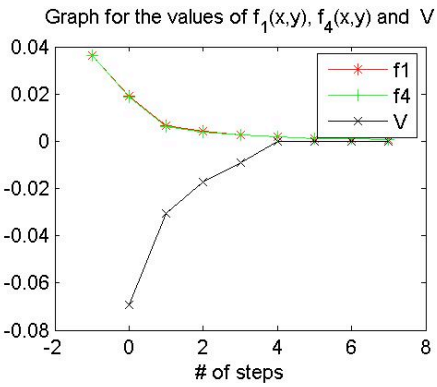
Primal Graph

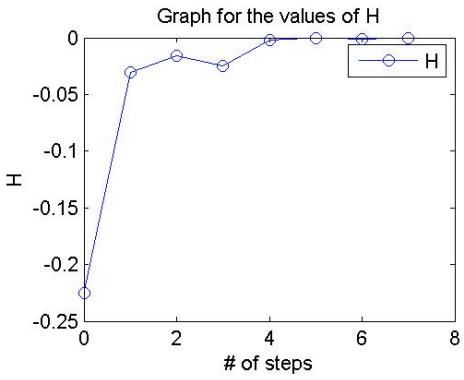


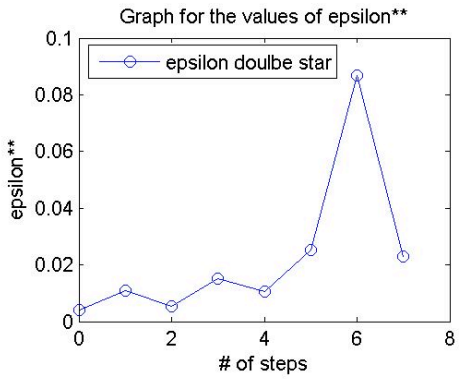




Dual Graph







Πίνακες Αποτελεσμάτων Συνολικών Πειραμάτων

Matrices R and C for win-loose Game **Primal Problem**					
	Παράδειγμα 1	**Παράδειγμα 2**	**Παράδειγμα 3**		
Ισορροπία Nash	0.001728	0.002023	0.002023		
Αρχική τιμή της f(x,y)	0.090700	0.107800	0.107800		
Αριθμός Βημάτων	7	15	15		
Αρχικά x και y	Uniform	Uniform	Uniform		

Matrices R and C for win-loose Game **Dual Problem**					
	Παράδειγμα 1	**Παράδειγμα 2**	**Παράδειγμα 3**		
Ισορροπία Nash	0.000665	0.001366	0.001366		
Αρχική	0.036438	0.041922	0.041922		

τιμή της f(x,y)					
Αριθμός Βημάτων	8	8	8		

Σημείωση: Οι αντίστοιχες γραφικές παραστάσεις και τα αρχεία εξόδου για τα υπόλοιπα παραδείγματα φαίνονται στα αρχεία κάτω από τον φάκελο / win-loose_second

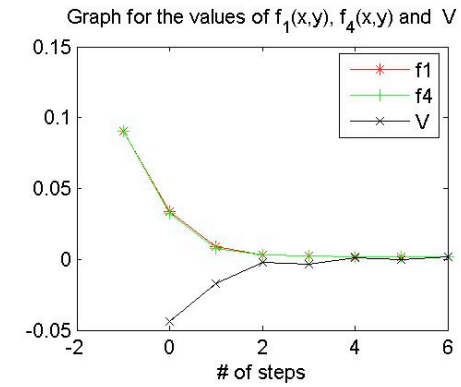
6.5.9 R και C πίνακες μεγέθους 100 x 100

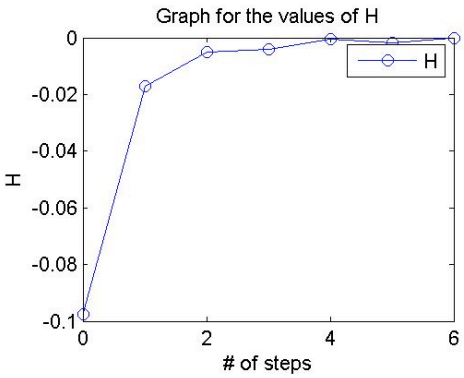
Χαρακτηριστικά Παιγνίου:

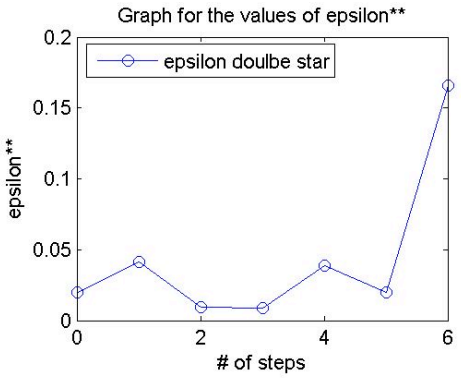
- Τύπος Παιγνίου: Win-loose παίγνιο
- Μέγεθος παιγνίου: 100 x 100
- Κριτήριο τερματισμού: $V(x,y)-f(x,y)\geq -\delta$
- Τιμή της παραμέτρου δ: 10^{-3}
- Αρχικές τιμές των x και y: Αγνές Στρατηγικές

Παράδειγμα 1 [αρχείο: *win-loose_second \100\1*]

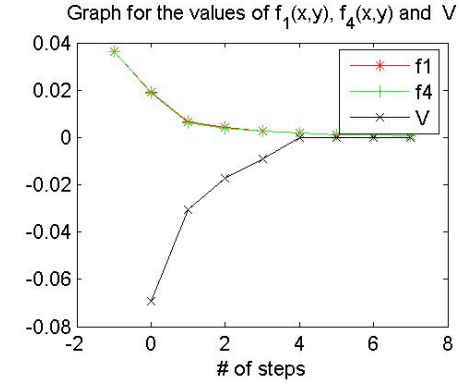
Primal Graph

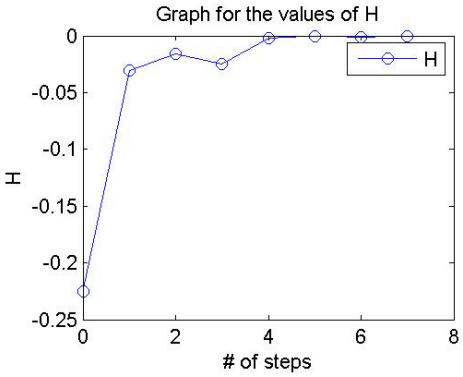


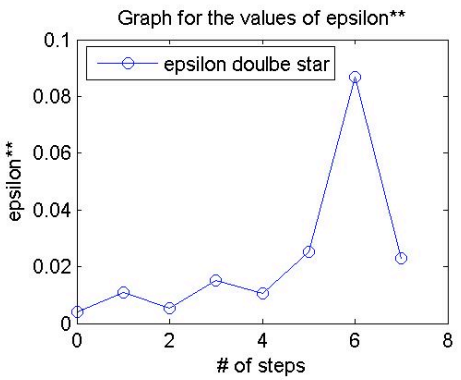




Dual Graph







Πίνακες Αποτελεσμάτων Συνολικών Πειραμάτων

Matrices R and C for win-loose Game **Primal Problem**					
	Παράδειγμα 1	**Παράδειγμα 2**	**Παράδειγμα 3**		
Ισορροπία Nash	0.001454	0.000965	0.003218		
Αρχική τιμή της f(x,y)	1.000000	1.000000	1.000000		
Αριθμός Βημάτων	17	15	9		
Αρχικά x και y	Αγνή στρατηγική	Αγνή στρατηγική	Αγνή στρατηγική		

Matrices R and C for win-loose Game **Dual Problem**					
	Παράδειγμα 1	**Παράδειγμα 2**	**Παράδειγμα 3**		
Ισορροπία Nash	0.001142	0.000846	0.001642		
Αρχική τιμή της f(x,y)	0.033173	0.045960	0.125594		
Αριθμός Βημάτων	17	8	9		

Σημείωση: Οι αντίστοιχες γραφικές παραστάσεις και τα αρχεία εξόδου για τα υπόλοιπα παραδείγματα φαίνονται στα αρχεία κάτω από τον φάκελο / win-loose_second

Παρατηρήσεις πάνω στα win-loose παίγνια:

- Αρχικά βλέπουμε ότι κι εδώ έχουμε πολύ καλές ισορροπίες Nash.
  Παρ` όλα αυτά βλέπουμε ότι το συγκεκριμένο παίγνιο είναι κάπως δυσκολότερο από το προηγούμενο.
  Δηλαδή υπήρχαν περιπτώσεις που έδωσε και ισορροπία της τάξης του 10^{-2}.
  Ακόμα και σε μικρού μεγέθους παίγνια υπήρχε μια δυσκολία εύρεσης καλών ισορροπιών Nash.
- Όταν στην primal λύση έχουμε αποτέλεσμα της τάξης του δ και μεγαλύτερης, τότε το dual δίνει καλύτερη λύση.
- Επίσης ο αριθμός των βημάτων μέχρι να τερματίσει ο αλγόριθμος είναι μεγαλύτερος από προηγούμενα προβλήματα και κυμαίνεται από 6 έως και 18 βήματα περίπου, ενώ για το dual πρόβλημα είναι συνήθως λιγότερα.
  Όταν χρησιμοποιήσαμε το δεύτερο κριτήριο τερματισμού τα βήματα είναι σαφώς περισσότερα. Στα συγκεκριμένα παίγνια εμφανίζεται ποιοτική διαφορά στην ισορροπία μόνο εκεί όπου είχαμε μεγάλη τελική τιμή της f(x,y).
  Παρ` όλα αυτά παρατηρήσαμε ότι τα περισσότερα από αυτά τα βήματα δεν αλλάζουν ουσιαστικά την τιμή της συνάρτησης f(x,y). Πράγμα που σημαίνει ότι με καλύτερη επιλογή του ε, ενδεχομένως αυτά τα βήματα να μην υπήρχαν.

Επιπλέον παρατηρήσεις στο τέλος του κεφαλαίου

6.6 Πίνακες κέρδους R και C για win-loose παίγνια με κύκλους (Matrices R και C for win-lose games with cycles)

Στην προσπάθεια μας να δημιουργήσουμε δύσκολα παίγνια για τον καλύτερο έλεγχο του προγράμματός μας, δημιουργήσαμε win-loose παίγνια που περιέχουν κύκλους.

Χαρακτηριστικά παιγνίων:

- Οι πίνακες κέρδους R και C, είναι πίνακες μεγέθους nxn και περιέχουν μόνο τιμές 0 ή 1.
- Για παίγνια μεγέθους 100x100 θα προτιμήσουμε να δημιουργήσουμε τέσσερις κύκλους απροσδιόριστα μεγάλους [έγινε προσπάθεια το πλήθος των 1 να είναι κοντά στο $n^2/3$], ενώ για μικρότερα προβλήματα, για παράδειγμα για 10x10 παίγνια, προτιμήσαμε να βάλουμε λιγότερους κύκλους (2 κύκλους).
  Ουσιαστικά προσπαθούμε να βάλουμε λίγους, αλλά μεγάλους κύκλους σε κάθε παίγνιο αυτού του είδους.
- Προσπαθούμε κατά την δημιουργία των κύκλων όλες οι στήλες του R να έχουν τουλάχιστον έναν "1" η κάθε μία και όλες οι γραμμές του C να έχουν τουλάχιστον έναν "1" η κάθε μία. Γιατί αν δεν συμβαίνει αυτό, τότε μπορεί σε κάθε περίπτωση να έχουμε αγνή exact ισορροπία Nash, κάτι το οποίο δεν θέλουμε.
  Η περίπτωση να έχουμε στην ίδια στήλη του R και C μόνο 0 ή στην ίδια γραμμή του R και C μόνο 0, δεν μας πειράζει διότι τελικώς αυτή η στήλη (ή γραμμή αντίστοιχα) δεν θα παιχτεί ποτέ από τους παίκτες.

Παρακάτω περιγράφεται η διαδικασία δημιουργίας κύκλων.

6.6.1 Περιγραφή δημιουργία κύκλων

Κύκλος

Κύκλο θα ονομάζουμε οποιαδήποτε ακολουθία σημείων $((i_1, j_1),(i_2, j_2),...,(i_k, j_k),(i_1, j_1))$ στους πίνακες R και C για τους οποίους πρέπει να ισχύουν οι εξής κανόνες:

- Το παίγνιο είναι win-lose. Δηλαδή οι πίνακες κέρδους R και C περιέχουν μόνο τιμές 0 και 1.
- Για τις τιμές των R και C ισχύει το εξής:
  Όταν σε μια θέση (i, j) του πίνακα R (ή αντίστοιχα του C) έχουμε την τιμή 0, τότε στην αντίστοιχη θέση (i, j) του πίνακα C (ή αντίστοιχα του R) μπορεί να υπάρχει είτε η τιμή 0, είτε η τιμή 1.
  Όταν σε μια θέση (i, j) του πίνακα R (ή αντίστοιχα του C) έχουμε την τιμή 1, τότε στην αντίστοιχη θέση (i, j) του πίνακα C (ή αντίστοιχα του R) μπορεί να υπάρχει μόνο η τιμή 0.
- Αν σε κάποιο σημείο (i_g, j_g) του κύκλου στον πίνακα R έχουμε την τιμή 0 και στον πίνακα C την τιμή 1 [δηλαδή $R_{(i_g,j_g)} = 0$ και $C_{(i_g,j_g)} = 1$], τότε στο αμέσως επόμενο σημείο (i_{g+1}, j_{g+1}) του κύκλου πρέπει η τιμή στον πίνακα R να είναι 1 και η τιμή στον πίνακα C να είναι 0 [δηλαδή: $R_{(i_{g+1},j_{g+1})} = 1$ και $C_{(i_{g+1},j_{g+1})} = 0$].
- Αν σε κάποιο σημείο (i_g, j_g) του κύκλου στον πίνακα R έχουμε την τιμή 1 και στον πίνακα C την τιμή 0 [δηλαδή: $R_{(i_g,j_g)} = 1$ και $C_{(i_g,j_g)} = 0$], τότε στο

αμέσως επόμενο σημείο του κύκλου πρέπει η τιμή στον πίνακα R να είναι 0 και η τιμή στον πίνακα C να είναι 1 [δηλαδή: $R_{(i_{g+1},j_{g+1})}=0$ και $C_{(i_{g+1},j_{g+1})}=1$].

- Ο αριθμός των σημείων που έχει ο κύκλος είναι πάντα άρτιος και η ακολουθία των σημείων ξεκινάει από ένα σημείο και τελειώνει στο ίδιο αυτό σημείο.

Σχηματικά:
Παράδειγμα ενός κύκλους που περιέχει μόνο 4 σημεία, σε ένα 5x5 πίνακα που αναπαριστά τις τιμές του R (πρώτη τιμή σε κάθε θέση) και C (δεύτερη τιμή σε κάθε θέση) δίνεται παρακάτω:

Πίνακες (R,C)				
(0,0)	(0,0)	(0,0)	(0,0)	(0,0)
(0,0)	**(0,1)**	(0,0)	**(1,0)**	(0,0)
(0,0)	(0,0)	(0,0)	(0,0)	(0,0)
(0,0)	**(1,0)**	(0,0)	**(0,1)**	(0,0)
(0,0)	(0,0)	(0,0)	(0,0)	(0,0)

Περιγραφή Δημιουργίας Κύκλων
Η δημιουργία των κύκλων στο πρόγραμμά μας θα έχει τα εξής χαρακτηριστικά:

- Θεωρούμε ότι οι πίνακες κέρδους R και C είναι αρχικά μηδενικοί. Έτσι στα μόνο σημεία που μπορούμε να έχουμε μη-μηδενική τιμή είναι σε αυτά που σχηματίζουν τους κύκλους, όπως περιγράψαμε παραπάνω.
- Θα προτιμήσουμε την δημιουργία λίγων κύκλων που ο καθένας περιέχει πολλά σημεία που τον σχηματίζουν.
- Όταν τελικά δημιουργηθούν όλοι οι κύκλοι πρέπει όλες οι στήλες του R να έχουν τουλάχιστον έναν "1" η κάθε μία και όλες οι γραμμές του C να έχουν τουλάχιστον έναν "1" η κάθε μία.
- Κατά την δημιουργία των κύκλων όταν θέτουμε τιμές σε κάποιο σημείο δεν πρέπει να καταστρέφονται οι τιμές που έχουν δημιουργηθεί στο σημείο αυτό από προηγούμενους ή από τον ίδιο τον κύκλο, γιατί με αυτόν τον τρόπο καταστρέφεται ο ίδιος ο κύκλος.
- Κάθε κύκλος θα έχει τουλάχιστον μέγεθος (δηλαδή αριθμό σημείων) τέσσερα.

Έστω ότι θέλουμε να δημιουργήσουμε k κύκλους με τους όλους τους κανόνες που αναφέραμε παραπάνω.

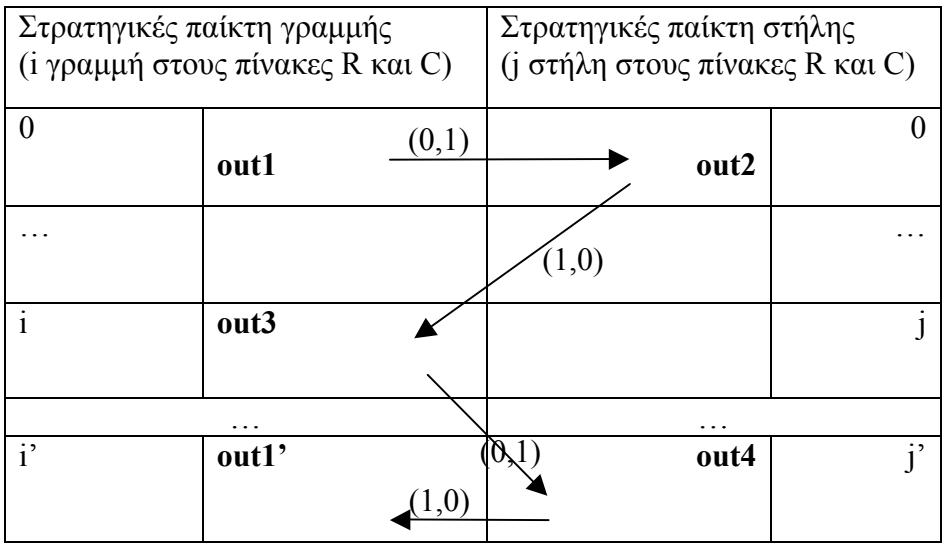
Τότε τους k-1 κύκλους τους δημιουργώ ως εξής:
Υποθέτουμε την ύπαρξη δύο συνόλων αριθμών. Το πρώτο σύνολο περιέχει τους αριθμούς (0,…,m), δηλαδή είναι ίσο με τον αριθμό των γραμμών των πινάκων κέρδους R και C (ουσιαστικά ίσο με το σύνολο των αγνών στρατηγικών του παίκτη γραμμής), ενώ το δεύτερο σύνολο περιέχει τους αριθμούς (0,…,n), δηλαδή είναι ίσο με τον αριθμό των στηλών των πινάκων κέρδους R και C (ουσιαστικά ίσο με το σύνολο των αγνών στρατηγικών του παίκτη στήλης).

1. Αρχικά επιλέγουμε τυχαία έναν αριθμό (out1) από το πρώτο σύνολο που θα αποτελεί και το αρχικό σημείο (start) του κύκλου αν είναι η πρώτη φορά που εκτελείται η επανάληψη.
   Αν δεν είναι η πρώτη φορά που εκτελείται η επανάληψη,
   τότε ελέγχουμε αν στη θέση (out1,out4),όπου out4 είναι το σημείο που επιλέξαμε στο βήμα 4, ο πίνακας C έχει τιμή 1.
   Αν όχι τότε θέτουμε στον πίνακα R στην θέση (out1,out4) την τιμή 1.
   Αν ναι τότε ξαναεκτελούμε το βήμα αυτό από την αρχή.
   Επίσης αν δεν είναι η πρώτη φορά που εκτελείται η επανάληψη,
   ελέγχουμε αν το σημείο out1 είναι ίδιο με το αρχικό (start) σημείο. Αν ναι τότε ο κύκλος έχει κλείσει και σταματάμε την επανάληψη.
2. Στη συνέχεια επιλέγουμε ένα τυχαίο νούμερο (out2) από το δεύτερο σύνολο.
   Ελέγχουμε αν στη θέση (out1,out2) ο πίνακας R έχει τιμή 1.
   Αν όχι τότε θέτουμε στον πίνακα C στην θέση (out1,out2) την τιμή 1.
   Αν ναι τότε ξαναεκτελούμε το βήμα αυτό από την αρχή.
3. Επιλέγουμε ένα τυχαίο νούμερο (out3) από το πρώτο σύνολο.
   Ελέγχουμε αν στη θέση (out3,out2) ο πίνακας C έχει τιμή 1.
   Αν όχι τότε θέτουμε στον πίνακα R στην θέση (out3,out2) την τιμή 1.
   Αν ναι τότε ξαναεκτελούμε το βήμα αυτό από την αρχή.
4. Επιλέγουμε ένα τυχαίο νούμερο (out4) από το δεύτερο σύνολο.
   Ελέγχουμε αν στη θέση (out3,out4) ο πίνακας R έχει τιμή 1.
   Αν όχι τότε θέτουμε στον πίνακα C στην θέση (out3,out4) την τιμή 1.
   Αν ναι τότε ξαναεκτελούμε το βήμα αυτό από την αρχή.

<u>Σημείωση:</u> Παρατηρούμε ότι κατά την δημιουργία των κύκλων για να μεταβούμε από το ένα σημείο στο επόμενο είτε διατηρούμε τον αριθμό της στήλης σταθερό, είτε τον αριθμό της γραμμής, εναλλάξ.

<u>Σχηματικά οι τιμές στην εκτέλεση μιας επανάληψης:</u>
Παρακάτω θα δούμε σχηματικά την επιλογή των πρώτων τεσσάρων σημείων εν΄ςο κύκλου.
Αυτά όπως είδαμε παραπάνω είναι κατά σειρά τα:

- (out1, out2)
- (out3, out2)
- (out3, out4)
- (out1', out4)



…			…
m			n

Τελευταίος Κύκλος

Λόγω του γεγονότος ότι θέλουμε όλες οι στήλες του R να έχουν τουλάχιστον έναν "1" η κάθε μία και όλες οι γραμμές του C να έχουν τουλάχιστον έναν "1" η κάθε μία., τον τελευταίο κύκλο τον δημιουργούμε ως εξής:
Ακολουθείται ο ίδιος αλγόριθμος με παραπάνω μόνο που τα out1, out2, out3 και out4 νούμερα δεν επιλέγονται τυχαία.
Δημιουργώντας τους k-1 πρώτους κύκλους, έχουμε κρατήσει σε διανύσματα τις στήλες του R που έχουν τουλάχιστον έναν 1 και τις γραμμές του C που έχουν τουλάχιστον έναν 1.
Έτσι στον τελευταίο κύκλο επιλέγουμε εκείνες τις στήλες (νούμερα του δεύτερου συνόλου) και εκείνες τις γραμμές (νούμερα του πρώτου συνόλου) στα οποία δεν υπάρχει κανένας 1 στους πίνακες R και C αντίστοιχα.
Επειδή με την επιλογή μη τυχαίων αριθμών υπάρχει η πιθανότητα να εμφανιστούν παραβάσεις των παραπάνω κανόνων (να υπάρξουν σημεία που χαλάνε κάποιον άλλο κύκλο που είχε δημιουργηθεί), κάνουμε ελέγχους και αν συμβεί κάτι τέτοιο απλώς αντί για το μη-τυχαίο σημείο επιλέγουμε ένα τυχαίο και αφήνουμε το μη-τυχαίο σημείο να επιλεγεί σε κάποια από τις επόμενες επαναλήψεις του αλγορίθμου.
Με άλλα λόγια, ο τελευταίος κύκλος καλύπτει τουλάχιστον τα σημεία των πινάκων R και C που δεν είχαμε 1 στις στήλες ή στις γραμμές αυτών, αντίστοιχα.

Οι πίνακες R και C για το συγκεκριμένο παίγνιο win-loose που περιγράψαμε παραπάνω, παράγονται μέσω κώδικα σε ANSI C, στο κυρίως πρόγραμμα.

Ο κώδικας είναι ο παρακάτω:

```
/* Generate matrices with cycles */

        for(i=0 ; i<m ;i++)
        {
                for(j=0 ; j<n ; j++)
                {
                        R_matrix_fixed[i][j] = 0.0;
                        C_matrix_fixed[i][j] = 0.0;
                }
        }

        // Create matrices for out1 and out3
        for(i=0 ; i<m ;i++)
        {
                //Counts the rows of C that has a 1
                out1_matrix[i]=i;
```

```
}
for(i=0 ; i<n ;i++)
{
	//Counts the cols of R that has a 1
	out4_matrix[i]=i;
}


for(k=0 ; k<1 ; k++)
{
 // FILE information - Start //
 fprintf(cycles_info, "Cycle: %d\n", k);
 // FILE information - End //
 while(1)
 {
	CYCLE1:
	out1 = (int) (rand() % m);


	if( flag_cycle == 0 )
	{
	   flag_cycle = 1;
	   start = out1;
	}
	else if(R_matrix_fixed[out1][out4] == 1.0)
	{
	   goto CYCLE1;
	}
	else
	{
	   R_matrix_fixed[out1][out4] = 0.0;
	   C_matrix_fixed[out1][out4] = 1.0;

	   out1_matrix[out1] = -1;

	   // FILE information - Start //
	   fprintf(cycles_info, "(%d,%d)\n", out1, out4);
	   // FILE information - End //

	   if(out1 == start)
	   {
		break;
	   }
	}

	CYCLE2:
	out2 = (int) (rand() % n);

	if(C_matrix_fixed[out1][out2] == 1.0)
	{
```

```
            goto CYCLE2;
        }
        else
        {
            R_matrix_fixed[out1][out2] = 1.0;
            C_matrix_fixed[out1][out2] = 0.0;

            out4_matrix[out2] = -1;

            // FILE information - Start //
            fprintf(cycles_info, "(%d,%d)\n", out1, out2);
            // FILE information - End //
        }

        CYCLE3:
        out3 = (int) (rand() % m);


        if(R_matrix_fixed[out3][out2] == 1.0)
        {
            goto CYCLE3;
        }
        else
        {
            R_matrix_fixed[out3][out2] = 0.0;
            C_matrix_fixed[out3][out2] = 1.0;

            out1_matrix[out3] = -1;

            // FILE information - Start //
            fprintf(cycles_info, "(%d,%d)\n", out3, out2);
            // FILE information - End //

            if(out3 == start)
            {
                break;
            }
        }

        CYCLE4:
        out4 = (int) (rand() % n);

        if(C_matrix_fixed[out3][out4] == 1.0)
        {
            goto CYCLE4;
        }
        else
        {
            R_matrix_fixed[out3][out4] = 1.0;
            C_matrix_fixed[out3][out4] = 0.0;
```

```
            out4_matrix[out4] = -1;

            // FILE information - Start //
            fprintf(cycles_info, "(%d,%d)\n", out3, out4);
            // FILE information - End //
        }
 } //end of while-cycles
flag_cycle = 0;
}// end for-cycles


// Last Cycle

// FILE information - Start //
fprintf(cycles_info, "Last Cycle\n");
for(i=0 ; i<m ;i++)
{
        fprintf(cycles_info, "%d\t", out1_matrix[i]);
}
fprintf(cycles_info, "\n");
for(i=0 ; i<n ;i++)
{
        fprintf(cycles_info, "%d\t", out4_matrix[i]);
}
fprintf(cycles_info, "\nLast Cycle:\n");
// FILE information - End //


while(1)
{
        CYCLE5:
        for(i=0 ; i<m ; i++)
        {
                if(out1_matrix[i]!= -1)
                {
                        out1 = i;
                        break;
                }
                else if(i==(m-1))
                {
                    out1 = (int) (rand() % m);
                    flag_out1 = 1;
                }
        }

        CYCLE5b:

        if( flag_cycle == 0 )
        {
```

```
    flag_cycle = 1;
    start = out1;
}
else if(R_matrix_fixed[out1][out4] == 1.0)
{
     out1 = (int) (rand() % m);
    goto CYCLE5b;
}
else
{
    R_matrix_fixed[out1][out4] = 0.0;
    C_matrix_fixed[out1][out4] = 1.0;

    out1_matrix[out1] = -1;

    // FILE information - Start //
    fprintf(cycles_info, "(out1, out4) = (%d,%d)\n", out1, out4);
    // FILE information - End //

    if(out1 == start)
    {
          break;
    }
}
CYCLE6:
out2 = (int) (rand() % n);
if(C_matrix_fixed[out1][out2] == 1.0)
{
    goto CYCLE6;
}
else
{
    R_matrix_fixed[out1][out2] = 1.0;
    C_matrix_fixed[out1][out2] = 0.0;

    // FILE information - Start //
    fprintf(cycles_info, "(out1, out2) = (%d,%d)\n", out1, out2);
    // FILE information - End //

    out4_matrix[out2] = -1;
}


CYCLE7:
out3 = (int) (rand() % m);
if(R_matrix_fixed[out3][out2] == 1.0 || out3 == start)
{
    goto CYCLE7;
}
else
```

```
            {
               R_matrix_fixed[out3][out2] = 0.0;
               C_matrix_fixed[out3][out2] = 1.0;

               // FILE information - Start //
               fprintf(cycles_info, "(out3, out2) = (%d,%d)\n", out3, out2);
               // FILE information - End //

               out1_matrix[out1] = -1;

               if(out3 == start)
               {
                     break;
               }
            }
            CYCLE8:
            for(i=0 ; i<n ; i++)
            {
                  if(out4_matrix[i]!= -1)
                  {
                        out4 = i;
                        break;
                  }
                  else if(i==(n-1))
                  {
                      out4 = (int) (rand() % n);
                      flag_out4 = 1;
                  }
            }

            CYCLE8b:
            if(C_matrix_fixed[out3][out4] == 1.0)
            {
               goto CYCLE7;
            }
            else
            {
               R_matrix_fixed[out3][out4] = 1.0;
               C_matrix_fixed[out3][out4] = 0.0;

               // FILE information - Start //
               fprintf(cycles_info, "(out3, out4) = (%d,%d)\n", out3, out4);
               // FILE information - End //

               out4_matrix[out4]= -1;
            }

      } //end of last cycle

      // FILE information - Start //
```

```
        fprintf(cycles_info, "\n");
        for(i=0 ; i<m ;i++)
        {
              fprintf(cycles_info, "%d\t", out1_matrix[i]);
        }
        fprintf(cycles_info, "\n");
        for(i=0 ; i<n ;i++)
        {
              fprintf(cycles_info, "%d\t", out4_matrix[i]);
        }
        // FILE information - End //
```

6.6.2 Ανάλυση 10x10 δύσκολου win-loose παιγνίου με 2 κύκλους (10x10 Win-Loose game, with two cycles)

Πριν παρουσιάσουμε διάφορα πειράματα για τέτοιου είδους παίγνια, θα παρουσιάσουμε αναλυτικά τα αποτελέσματα πάνω σε ένα 10x10 δύσκολου win-loose παιγνίου με 2 κύκλους.
Σε αυτό θα φανεί η όλη διαδικασία δημιουργίας αυτού του είδους των δύσκολων παιγνίων, αλλά θα γίνει μια σύντομη παρουσίαση των αποτελεσμάτων στο πιο δύσκολο παίγνιο που εμφανίστηκε μεταξύ όλων όσων εκτελέσαμε μέχρι στιγμής.

Πίνακας Κέρδους (R,C) των δύο παικτών:
Αρχικά μπορούμε να δούμε τον πίνακα που δημιουργήθηκε εκτελώντας τον παραπάνω κώδικα για 10x10 μεγέθους πίνακα για δύο κύκλους.
Η πρώτη τιμή εκφράζει το κέρδος του παίκτη γραμμής (ουσιαστικά είναι ο πίνακας κέρδους R) και η δεύτερη το κέρδος του παίκτη στήλης (ουσιαστικά είναι ο πίνακας κέρδους C).
Παρακάτω παρουσιάζονται ταυτόχρονα οι δύο κύκλοι που δημιουργήθηκαν, καθώς και η πορεία δημιουργίας τους (που εκφράζεται με βέλη).
Ο ένας κύκλος παρουσιάζεται με κόκκινη γραμμή και ο άλλος με πράσινη γραμμή.

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	(0,0)	**(0,1)**	(0,0)	**(1,0)**	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)
1	**(0,1)**	(0,0)	(0,0)	(0,0)	**(1,0)**	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)
2	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	**(1,0)**	**(0,1)**
3	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	**(0,1)**	(0,0)	**(1,0)**	(0,0)	(0,0)
4	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)
5	(0,0)	(0,0)	(0,0)	**(0,1)**	(0,0)	**(0,1)**	(0,0)	**(1,0)**	(0,0)	**(1,0)**
6	**(1,0)**	(0,0)	**(0,1)**	(0,0)	(0,0)	**(1,0)**	(0,0)	**(0,1)**	(0,0)	(0,0)
7	(0,0)	(0,0)	**(1,0)**	(0,0)	(0,0)	(0,0)	**(0,1)**	(0,0)	(0,0)	(0,0)
8	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	**(1,0)**	(0,0)	**(0,1)**	(0,0)

9	(0,0)	**(1,0)**	(0,0)	(0,0)	**(0,1)**	**(1,0)**	(0,0)	**(0,1)**	(0,0)	(0,0)

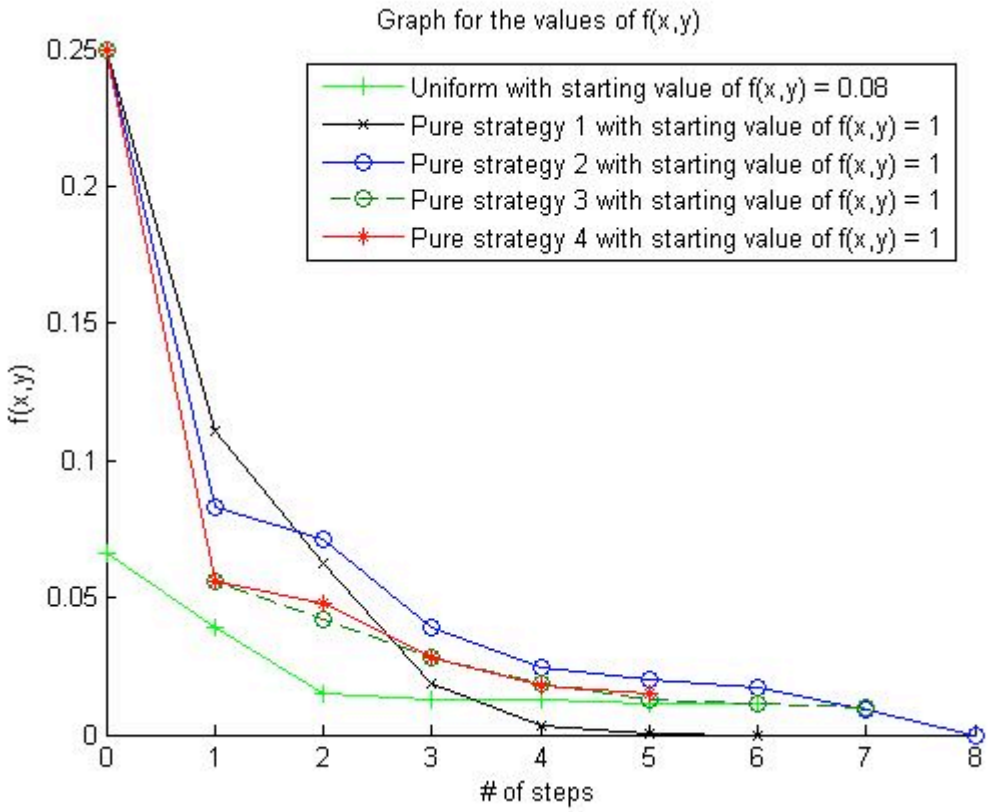
Τα κατά σειρά σημεία που δημιούργησαν τους δύο κύκλους είναι τα εξής:

Πρώτος Κύκλος:	**Δεύτερος Κύκλος:**
(7,6) (8,6) (8,8) (2,8) (2,9) (5,9) (5,3) (0,3) (0,1) (9,1) (9,4) (1,4) (1,0) (6,0) (6,2) (7,2)	(3,5) (9,5) (9,7) (5,7) (5,5) (6,5) (6,7) (3,7)

PRIMAL Πρόβλημα

Παράθεση δεδομένων εκτέλεσης του πειράματος

Αρχικά παρουσιάζουμε την γραφική παράσταση που δημιουργήθηκε με τον δεύτερο κώδικα που περιγράψαμε παραπάνω. Οι καμπύλες που εμφανίζονται παρουσιάζουν τις τιμές της συνάρτησης f(x,y), για διάφορα starting points x και y, τα οποία περιγράφονται σε εσωτερική λεζάντα.



Πίνακες Αποτελεσμάτων

Στους πίνακες παρακάτω παρουσιάζονται:

- οι αρχικές τιμές της συνάρτησης f(x,y)
- η προσεγγιστική ισορροπία Nash
- ο αριθμός των βημάτων εκτέλεσης του παιγνίου από 5 διαφορετικά starting points.
- Οι αρχικές πιθανοτικές κατανομές x και y.

Σημείωση: Οι pure strategies διαλέχτηκαν έτσι ώστε αρχική τιμή της f να είναι 1

Matrices R and C for win-loose Game with cycles **Primal Problem**					
	Uniform	**Pure Strategy 1**	**Pure Strategy 2**	**Pure Strategy 3**	**Pure Strategy 4**
N.E.	0.010989	0.000000	0.000000	0.0095238	0.0151515
Αρχική τιμή της f	0.080000	1.000000	1.000000	1.000000	1.000000
Αριθμός Βημάτων	7	7	9	8	6
Starting points x, y	$x_i = 0.1$ $y_j = 0.1$ για κάθε i, j	$x_0 = 1$ $y_0 = 1$	$x_3 = 1$ $y_0 = 1$	$x_5 = 1$ $y_0 = 1$	$x_7 = 1$ $y_0 = 1$

Πιθανοτικές κατανομές x και y

Παρακάτω παρουσιάζονται οι τιμές των πιθανοτικών κατανομών **x** και **y**, στο stationary point όπου βρέθηκε NE σε κάθε περίπτωση:

Uniform:									
x:									
0.153846	0.153846	0.153846	0	0	0.153846	0	0.076923	0.153846	0.15384
y:									
0.142857	0.142857	0	0.142857	0.142857	0	0.142857	0.142857	0.142857	0

Pure Strategy 1:									
x:									
0.250000	0.250000	0	0	0	0.250000	0	0	0	0.25000
y:									
0	0.250000	0	0.250000	0.250000	0	0	0.250000	0	0

Pure Strategy 2:									
x:									
0	0	0.200000	0.200000	0	0	0.200000	0.200000	0.200000	0
y:									
0	0	0.200000	0	0	0.200000	0.200000	0.200000	0.200000	0

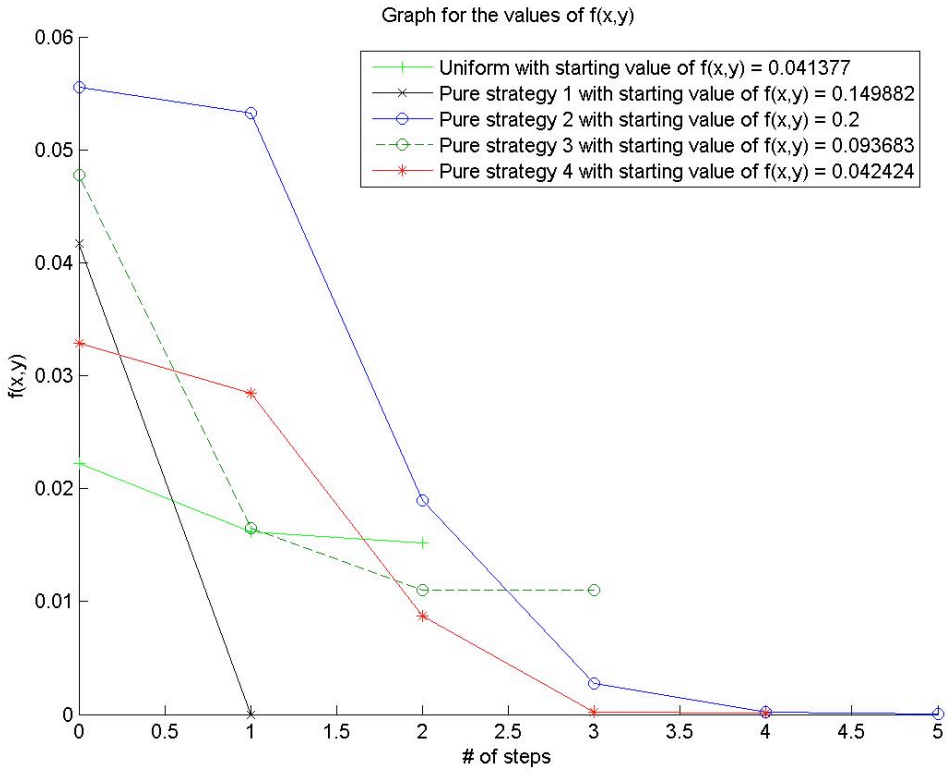
Pure Strategy 3:									
x:									
0.142857	0.142857	0.142857	0	0	0.142857	0.142857	0.142857	0.142857	0
y:									
0.133333	0.133333	0.133333	0.133333	0.066667	0	0.133333	0.133333	0.133333	0

Pure Strategy 4:									
x:									
0	0.166667	0.166667	0	0	0.166667	0.166667	0.166667	0.166667	0
y:									
0.181818	0	0.181818	0	0.090909	0	0.181818	0.181818	0.181818	0

DUAL Πρόβλημα

Ομοίως παρουσιάζονται και τα αποτελέσματα για το dual πρόβλημα.

Παράθεση δεδομένων εκτέλεσης του πειράματος



Πίνακες Αποτελεσμάτων

Matrices R and C for win-loose Game with cycles **Dual Problem**					
	Uniform	**Pure Strategy 1**	**Pure Strategy 2**	**Pure Strategy 3**	**Pure Strategy 4**
N.E.	0.0151515	0.000000	0.0000017	0.010989	0.0001319
Αρχική τιμή της f	0.041377	0.149882	0.200000	0.093683	0.042424
Αριθμός Βημάτων	3	2	6	4	5

Πιθανοτικές κατανομές x και y

Uniform:

x:									
0.181818	0.181818	0.181818	0	0	0.181818	0	0	0.090909	0.18181
y:									
0.166667	0.166667	0	0.166667	0.166667	0	0	0.166667	0.166667	0

Pure Strategy 1:									
x:									
0.333333	0	0	0	0	0.333333	0	0	0	0.33333
y:									
0	0	0	0.266667	0.200000	0.266667	0	0.266667	0	0

Pure Strategy 2:									
x:									
0.166668	0	0.166658	0	0	0.166668	0.166668	0.166668	0.166668	0
y:									
0	0	0.166667	0.166667	0	0.166667	0.166667	0.000456	0.166667	0.16621

Pure Strategy 3:									
x:									
0.153846	0.153846	0.153846	0	0	0.153846	0	0.076923	0.153846	0.15384
y:									
0.142857	0.142857	0	0.142857	0.142857	0	0.142857	0.040179	0.142857	0.10267

Pure Strategy 4:									
x:									
0.166799	0	0.166799	0	0	0.166799	0.166004	0.166799	0.166799	0
y:									
0	0	0.166007	0.166799	0	0.166799	0.166799	0	0.166799	0.16679

Παρατηρήσεις πάνω στο win-loose παίγνιο με 2 κύκλους:

- Αρχικά να πούμε ότι το συγκεκριμένο παίγνιο αποτελεί μια εξαίρεση στις συνηθισμένες λύσεις που παρατηρήθηκαν σε 10x10 μεγέθους παίγνια. Σε μικρά παίγνια κατά 99% ο αλγόριθμος έδωσε exact ισορροπία Nash.
- Όμως όπως βλέπουμε και στο συγκεκριμένο παράδειγμα μερικές φορές εγκλωβίζεται σε μεγάλα supports και σταματάει σε stationary point που δίνει μεγάλη τιμή για την f(x,y)
- Παρ` όλα αυτά, αυτό που παρατηρήθηκε είναι ότι το ίδιο πρόβλημα από διαφορετικές γωνίες και διαφορετικά starting points δίνει σίγουρα κάποια ισορροπία Nash της τάξης του δ ή και μικρότερη.
- Φυσικά ενδέχεται, όπως βλέπουμε και παραπάνω, να δώσει ισορροπία Nash μεγαλύτερη από κάποιο άλλο αρχικό σημείο.
- Επίσης παρατηρούμε ότι δεν υπάρχει πρόβλημα για μικρά supports. Είναι όλα μεγαλύτερα από n/2 και διαφορετικά μεταξύ τους, όταν τρέχουμε τον αλγόριθμο από διαφορετικά starting points.

Επιπλέον παρατηρήσεις στο τέλος του κεφαλαίου

6.6.3 R και C πίνακες μεγέθους 100 x 100

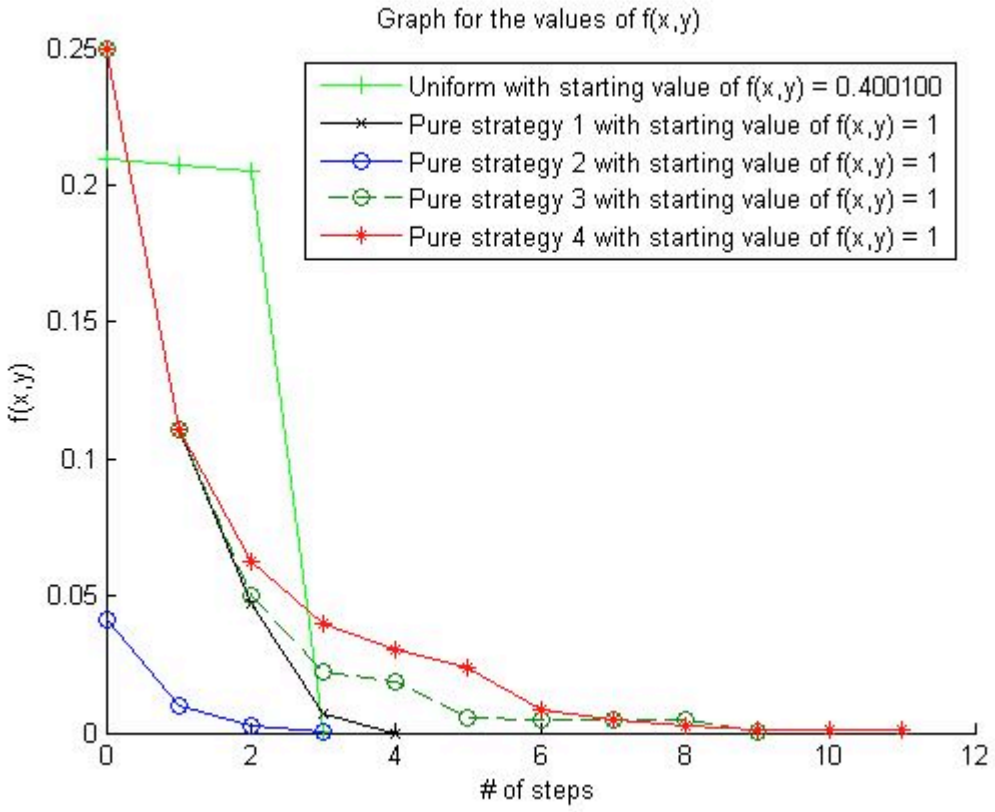
Η υποψία ότι τα μικρά win-loose παίγνια είναι δύσκολο να δημιουργήσουν δύσκολα παίγνια, μας οδήγησε στο να εκτελέσουμε πειράματα μόνο για μεγάλα παίγνια με λίγους (4) κύκλους. Αυτά παρουσιάζονται παρακάτω.

Χαρακτηριστικά Παιγνίου:

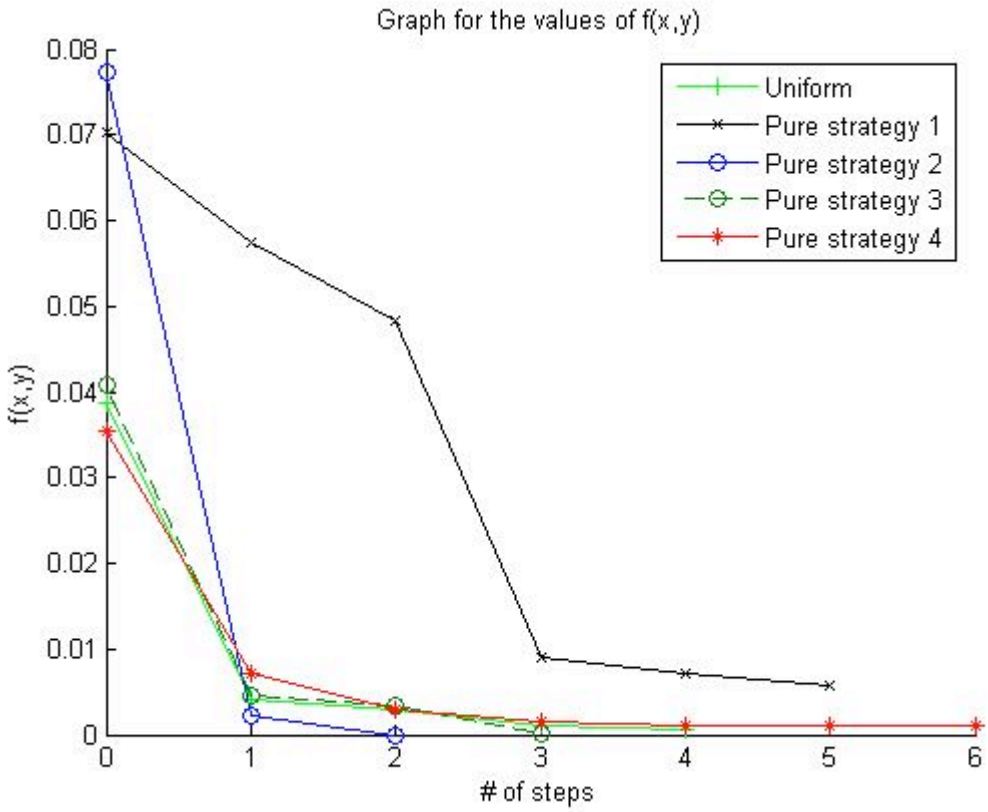
- Τύπος Παιγνίου: Win-loose παίγνιο με 4 κύκλους
- Μέγεθος παιγνίου: 100 x 100
- Κριτήριο τερματισμού: $f4 > f1 * [1-(\delta/(1+\delta))^2$
- Τιμή της παραμέτρου δ: 10^{-3}
- Αρχικές τιμές των x και y: Ομοιόμορφη κατανομή και Αγνές Στρατηγικές

Παράδειγμα 1 [αρχείο: *\cycles\100x100*]

Primal Graph



Dual Graph



Matrices R and C for win-loose Game with cycles **Primal Problem**					
	Παράδειγμα 1	**Παράδειγμα 2**	**Παράδειγμα 3**	**Παράδειγμα 4**	**Παράδειγμα 5**
N.E.	0.000651	0.000000	0.000279	0.000002	0.000857
Αρχική τιμή της f	0.400100	1.000000	1.000000	1.000000	1.000000
Αριθμός Βημάτων	4	5	4	10	12
Αρχικά x και y	Uniform	Αγνή Στρατηγική	Αγνή Στρατηγική	Αγνή Στρατηγική	Αγνή Στρατηγική

Matrices R and C for win-loose Game with cycles **Dual Problem**					
	Παράδειγμα 1	**Παράδειγμα 2**	**Παράδειγμα 3**	**Παράδειγμα 4**	**Παράδειγμα 5**
Ισορροπία Nash	0.000579	0.005718	0	0.000159	0.000974
Αρχική τιμή της f(x,y)	0.084266	0.162500	0.290292	0.163311	0.070132

Αριθμός Βημάτων	5	6	3	4	7

Σημείωση: Οι αντίστοιχες γραφικές παραστάσεις και τα αρχεία εξόδου για τα υπόλοιπα παραδείγματα φαίνονται στα αρχεία κάτω από τον φάκελο /cycles

Παρατηρήσεις πάνω στα win-loose παίγνια με κύκλους:

- Αρχικά βλέπουμε ότι κι εδώ έχουμε πολύ καλές ισορροπίες Nash.
  Όπως είπαμε και παραπάνω το συγκεκριμένο παίγνιο γίνεται δύσκολο για συγκεκριμένες περιπτώσεις.
  Όπως βλέπουμε εδώ όλες οι ισορροπίες είναι πολύ καλές και μικρότερες ή ίσες της τάξης του δ.
- Όταν στην primal λύση έχουμε αποτέλεσμα της τάξης του δ και μεγαλύτερης, τότε το dual δίνει καλύτερη λύση, σε λιγότερα συνήθως βήματα.
- Επίσης ο αριθμός των βημάτων μέχρι να τερματίσει ο αλγόριθμος κυμαίνεται από 3 έως και 11 βήματα περίπου χρησιμοποιώντας το δεύτερο κριτήριο τερματισμού, ενώ για το dual πρόβλημα είναι συνήθως λιγότερα (2 έως 6 βήματα).
  Παρ` όλα αυτά παρατηρήσαμε ότι τα περισσότερα από αυτά τα βήματα δεν αλλάζουν ουσιαστικά την τιμή της συνάρτησης f(x,y). Πράγμα που σημαίνει ότι με καλύτερη επιλογή του ε, ενδεχομένως αυτά τα βήματα να μην υπήρχαν.
- Αυτό που παρατηρήθηκε και εδώ είναι ότι το ίδιο πρόβλημα από διαφορετικές γωνίες και διαφορετικά starting points δίνει σίγουρα κάποιες διαφορετικές ισορροπίες Nash της τάξης του δ ή και μικρότερης, με διαφορετικά supports.
- Επίσης παρατηρούμε ότι δεν υπάρχει πρόβλημα για μικρά supports. Είναι όλα μεγαλύτερα από n/2 και διαφορετικά μεταξύ τους, όταν τρέχουμε τον αλγόριθμο από διαφορετικά starting points.

Επιπλέον παρατηρήσεις στο τέλος του κεφαλαίου

6.7 Γενικά Συμπεράσματα

Μετά από την παρουσίαση των πειραματικών αποτελεσμάτων και των επιμέρους συμπερασμάτων για κάθε ένα από τα είδη παιγνίων που δημιουργήσαμε, μπορούμε πλέον να βγάλουμε κάποια συνολικά συμπεράσματα που αφορούν τον αλγόριθμο που υλοποιήσαμε και περιγράφεται θεωρητικά στην δημοσιευμένη εργασία [1].

- Το πρώτο σημαντικό συμπέρασμα στο οποίο μπορούμε να καταλήξουμε τελικά είναι ότι ποιοτικά η προσεγγιστική ισορροπία Nash στην οποία συγκλίνει ο αλγόριθμος είναι πολύ καλύτερη από την χειρότερη περίπτωση που υπολογίστηκε θεωρητικά (0.3393).
  Στα πειράματα που έγιναν, για όλα τα είδη παιγνίων, οι προσεγγιστικές ισορροπίες Nash που βρέθηκαν, στα stationary points, ήταν της τάξης του δ, δηλαδή της τάξης του 10^{-3}, ή ακόμα και πολύ μικρότερες (πολλές φορές βρίσκαμε exact ισορροπία Nash).

  Επίσης παρατηρήσαμε ότι στα πειράματα που τρέξαμε η χειρότερη περίπτωση ήταν να βρεθεί stationary point που έδινε προσεγγιστική ισορροπία Nash της τάξης του 10^{-2}. Όμως στο ίδιο πρόβλημα από διαφορετικά starting points x και y, βρίσκαμε είτε exact ισορροπία Nash, είτε μικρότερης τάξης. Αυτό σημαίνει ότι σε όλα τα πειράματα μας βρέθηκε ισορροπία Nash, που ήταν της τάξης του δ και μικρότερα. Ουσιαστικά για την ακρίβεια που τα τρέξαμε τα θεωρούμε ως exact ισορροπίες Nash.

  Αυτό είναι ένα πολύ σημαντικό συμπέρασμα, γιατί βλέπουμε μια άριστη συμπεριφορά της συνάρτησης f(x,y) που δείχνει ότι μπορεί σε κάθε περίπτωση να υπολογιστεί μια πολύ καλή (της τάξης του 10^{-3} και μικρότερη) προσεγγιστική ισορροπία Nash σε πολυωνυμικό χρόνο και βήματα.

- Επίσης ο αλγόριθμος τερματίζει σε σχετικά μεγάλο support. Δηλαδή της τάξης μεγέθους από n/3 έως n/2, όπου n είναι το πλήθος των στρατηγικών των παικτών. Αυτό σημαίνει ότι δεν υπάρχει πρόβλημα εύρεσης ισορροπίας Nash με μικρό support.

- Επίσης μπορούμε να δούμε ότι σε αντίθεση με την θεωρητική ανάλυση που αναφέρει πολυωνυμικό πλήθος βημάτων για τον τερματισμό του αλγορίθμου, φαίνεται θεωρητικά ο αριθμός των βημάτων να είναι πολύ μικρότερος. Φαίνεται να είναι μικρότερο από $(1 / \delta^2)$, όπου δ είναι η παράμετρος ακρίβειας που ορίσαμε παραπάνω.
  [Αυτό αποτελεί μια σημαντική παρατήρηση όσο αφορά τη σύγκριση αυτού του αλγορίθμου με άλλους, διότι αναφέρθηκε στο παρελθόν ότι το αρνητικό του αλγορίθμου μας είναι ο πολυωνυμικός αριθμός προβλημάτων Γραμμικού Προγραμματισμού (δηλαδή βημάτων του αλγορίθμου) που έχουμε να λύσουμε. Κάτι το οποίο στην πράξη δεν φαίνεται να ισχύει.]

- Επίσης φαίνεται ότι ο αριθμός των βημάτων μέχρι τον τερματισμό του αλγορίθμου δεν αυξάνεται πολύ (για παράδειγμα εκθετικά) σε σχέση με το μέγεθος του προβλήματος.
  Όπως είδαμε με το πρώτο κριτήριο τερματισμού στα 100x100 προβλήματα

κάναμε περίπου 15 βήματα, ενώ με το δεύτερο κριτήριο κοντά στα 20 βήματα.

- Επιπλέον βλέπουμε ότι χρησιμοποιώντας, τις dual κανονικοποιημένες μεταβλητές που βρήκαμε στην τελευταία επανάληψη του αλγορίθμου (σε αυτό που ονομάσαμε primal πρόβλημα) ως αρχικές πιθανοτικές x και y, τότε φτάνουμε σε μια πολύ καλύτερη προσεγγιστική ισορροπία Nash και συνήθως σε λιγότερα βήματα.
  Ιδιαίτερα όταν η τιμή της τελικής f(x,y) στο primal πρόβλημα ήταν μεγαλύτερη του δ.

  Αυτό συμβαίνει διότι μέσω των dual μεταβλητών η επίλυση του προβλήματος δεν αρχίζει από τυχαία σημεία, αλλά από κάποια καλά σημεία του πολυέδρου που βρέθηκαν στην προηγούμενη εκτέλεση του αλγορίθμου (στο primal πρόβλημα).

  Το ανοικτό πρόβλημα εδώ είναι αν όντως μπορεί να αποδειχθεί ότι χρησιμοποιώντας πάντα (αναδρομικά) τις dual μεταβλητές του τελευταίου βήματος του αλγορίθμου, βρίσκουμε πάντα καλύτερες προσεγγιστικές ισορροπίες Nash. Και αν τελικά αυτό οδηγεί στην εύρεση μιας exact ισορροπίας Nash, αλλά και σε πόσα βήματα, δηλαδή αν αυτό γίνεται πολυωνυμικά ή όχι.

- Επίσης παρατηρούμε ότι ξεκινώντας από διαφορετικά αρχικά σημεία καταλήγουμε σε εξίσου καλές ισορροπίες Nash. Ενδεχομένως (όπως είδαμε στους κύκλους) μπορεί να καταλήξουμε ακόμα και σε καλύτερες ή χειρότερες ισορροπίες.
  Αλλά αυτό που παρατηρήσαμε είναι ότι υπάρχει τουλάχιστον κάποιο αρχικό σημείο (x,y) που να δίνει μια καλή ισορροπία Nash, της τάξης του δ και μικρότερη.

Κάποια επιμέρους συμπεράσματα που μπορούμε να εξάγουμε μέσω των γραφικών ή των αρχείων εξόδου του προγράμματος είναι ότι:

- Παρατηρούμε ότι η μεταβλητή **ε** (η ε** ουσιαστικά) ακολουθεί μια ακανόνιστη ακολουθία τιμών κατά την διάρκεια εκτέλεσης του αλγορίθμου.
  Αυτό είναι φυσιολογικό, διότι το μέγεθος των βημάτων από ένα σημείο του πολυέδρου σε κάποιο άλλο που επιτυγχάνει καλύτερη τιμή της f(x,y), δεν είναι και δεν θα μπορούσε να είναι σταθερό ή να έχει κάποιο πρότυπο (pattern), γιατί τότε η βέλτιστη λύση θα ήταν τετριμμένη και γνωστή.
- Η διαδικασία εύρεσης του **ε, ε* και ε**,** είναι τέτοιο που ενώ δίνει το βήμα προς ένα σημείο του πολυέδρου όπου εμφανίζεται μια καλύτερη τιμή για την f(x,y), ουσιαστικά μπορεί να μην δίνει την βέλτιστη.
  Η εύρεση του ε μέσω μιας σειριακής αναζήτησης πάνω σε όλα τα ε από 0 έως 1 για κάθε επανάληψη του τέταρτου βήματος του αλγορίθμου [ή ενδεχομένως πάνω σε όλα τα ε που απέχουν μεταξύ τους μια μικρή τιμή, ας πούμε 10^{-3}], θα έδινε σίγουρα την καλύτερη δυνατή τιμή για το ε.
  Αυτό σημαίνει ότι βρίσκοντας το βέλτιστο ε, ενδεχομένως να είχαμε πολύ καλύτερη τελική τιμή για την f(x,y), δηλαδή καλύτερη ποιοτικά προσεγγιστική ισορροπία Nash, αλλά και επίτευξη αυτού με λιγότερα βήματα.
- Επίσης κατά τον υπολογισμό του **ε** προσπαθήσαμε να μην περιλάβουμε πολλές αφαιρέσεις, ώστε να μην υπάρχουν πολλά σφάλματα κατά τον υπολογισμό του.
- Παρατηρούμε επίσης ότι πάντα η **τιμή της f(x,y)** μειώνεται μονότονα, είτε αναφερόμαστε στην f(x,y) που υπολογίζεται μετά το πρώτο βήμα, είτε σε αυτήν

που υπολογίζεται μετά το τέταρτο βήμα.
Αυτό είναι φυσιολογικό, διότι ο αλγόριθμος βρίσκει πάντα νέα σημεία του πολυέδρου που βρίσκουν σίγουρα καλύτερη τιμή για την f(x,y). Διαφορετικά σταματάει.

- Επίσης βλέπουμε ότι η **τιμή της f(x,y)** μετά το βήμα 4 είναι πάντα καλύτερη αυτής που υπολογίστηκε μετά το βήμα ένα στην ίδια επανάληψη του αλγορίθμου. [η καμπύλη της $f_4(x,y)$ είναι κάτω από αυτήν της $f_1(x,y)$]
  Και αυτό είναι φυσιολογικό, διότι, όπως είπαμε και παραπάνω, η επιλογή του ε έγινε με τέτοιο τρόπο ώστε η τιμή της συνάρτηση f(x,y) να μειώνεται μονότονα.
- Η μεταβλητή **V(x,y)-f(x,y)** εκφράζει την κλίση της συνάρτησης στο πολύεδρο, κατά την εύρεση κατεύθυνσης για το νέο σημείο (x,y).
  Με άλλα λόγια η μεταβλητή **V(x,y)** εκφράζει την descent κατεύθυνση της συνάρτησης f(x,y).
  Ακολουθώντας της κατεύθυνση αυτήν με το κατάλληλο ε, όπως είπαμε, βρίσκουμε νέο σημείο του πολυέδρου που δίνει καλύτερη τιμή στη συνάρτηση f(x,y).
  Η V(x,y),που αποτελεί την λύση του minimax προβλήματος, πρέπει να είναι πάντα μικρότερη της f(x,y), όπως και συμβαίνει και τελικά όταν συγκλίνουμε σε κάποιο stationary point, η τιμή της V(x,y) να προσεγγίζει την τιμή της f(x,y). Δηλαδή και οι δύο καμπύλες να πλησιάζουν τον άξονα x στις γραφικές.
  Όταν έχουμε exact ισορροπία Nash, τότε η τιμή της V(x,y) = 0
- Η μεταβλητή **H** εκφράζει την καμπυλότητα που έχουμε. Δηλαδή εκφράζει κάτι σαν την δεύτερη παράγωγο της συνάρτησης. Όταν είναι αρνητική σημαίνει ότι η καμπύλη είναι κυρτή.
  Παρατηρούμε ότι παρόλη την κάποιας μορφής ακανόνιστης συμπεριφοράς τελικά αυτή μειώνεται και προσεγγίζει το 0, όταν πλησιάζουμε σε ισορροπία Nash.
- Μιλήσαμε αρκετά για τις τιμές των μεταβλητών κατά την επίλυση του dual προβλήματος **w, z**. Αυτό που πρέπει να πούμε είναι ότι είναι πίνακες πιθανοτήτων που παίζουν τον ρόλο της πιθανοτικής κατανομής πάνω στους πίνακες κέρδους (ή κόστους ανάλογα) για να δείξουν την «τιμωρία» του να αποκλίνεις από τα supports $S_R(y)$ και $S_C(x)$

  Επίσης αξίζει να σημειώσουμε ότι οι dual μεταβλητές έχουν ίδιο πρόσημο γιατί βάλαμε ίδιας φοράς ανισότητες. Το ότι βγαίνουν όλες αρνητικές είναι συμβολικό, για αυτό παίρνουμε και τις απόλυτες τιμές για την χρήση τους ως πιθανοτικές κατανομές στην συνέχεια του προγράμματος.
- Η παράμετρος **ρ** είναι η παράμετρος εξισορρόπησης μεταξύ των δύο μερών ($f_R(x,y)$ και $f_C(x,y)$) της συνάρτησης f(x,y).
  Όπως βλέπουμε το άθροισμα των ρ και (1-ρ) είναι πάντα 1, όπως και θα έπρεπε.

  Κατά την διάρκεια της εκτέλεσης του αλγορίθμου, όταν βρίσκαμε το ρ = 0, τότε θέταμε τις θέσεις του w ισοπίθανες (ομοιόμορφη κατανομή) στις θέσεις που έδειχνε το SR(y), διότι δεν μας ενδιέφερε πως θα καθοριστούν οι πιθανότητες μιας και το πρόβλημα γίνονταν εκφυλισμένο πλέον.
- Όσο αφορά τις τιμές των μεταβλητών **λ και μ**, πρέπει να μην είναι αρνητικές όταν σταματάμε τον αλγόριθμο και έχουμε βρει κάποιοι stationary point.
  Αν τουλάχιστον ένα από τα δύο είναι αρνητικό, τότε η descent διαδικασία πρέπει να συνεχίσει.
  Στο σημείο που έχουμε ισορροπία Nash πρέπει να ισχύει:
  f(x,y) < [(λ*μ) / (λ+μ)]

Επίσης οι τιμές των λ και μ πρέπει να είναι πάντα μικρότερες του 1. Δηλαδή:
$\lambda, \mu \leq 1$
ενώ στα stationary points πρέπει να ισχύει:
$0 \leq \lambda, \mu \leq 1$

Τέλος παραθέτουμε κάποιες ανισότητες που ισχύουν όταν βρισκόμαστε σε κάποιο stationary point και θα μπορούσαν να λειτουργήσουν ως σημεία επαλήθευσης της ορθότητας του αλγορίθμου:

- $V(x,y) - f(x,y) \geq -\delta$
- $V(x,y) \leq \rho * \lambda$ και $V(x,y) \leq (1-\rho) * \mu$
- Ή ακόμα πιο ισχυρά:
  $V(x,y) \leq \min(\rho * \lambda, (1-\rho) * \mu)$
- Όταν έχουμε exact ισορροπία Nash:
  $V(x,y) = 0$ και λ, μ όχι αρνητικά

Βιβλιογραφία:

[1] Tsaknakis, H., Spirakis, P. G
An Optimization Approach for Approximate Nash Equilibria.
In: WINE 2007, LNCS 4858, pp. 42-56, 2007, Springer-Verlag, Berlin, Heilderberg (2007).

[2] Παύλος Σπυράκης και Λευτέρης Κυρούσης.
Σημειώσεις για το μάθημα «Οικονομική Θεωρία και Αλγόριθμοι»,

[3] John von Neumann and Oskar Morgenstern.
Theory of Games and Economic Behavior,
Princeton University Press (1944).

[4] J. Nash.
Equilibrium point in n-person games
Proceedings of the national Academy of the USA,
36(1):48-49, 1950

[5] J. Nash.
Noncooperative games
Annals of Mathematics,
54: 289-295, 1951

[6] D. Bertsimas, J. N. Tsitsiklis
Introduction to Linear optimization
MIT press, 1997

[7] Laurence A. Wolsey, George L. Nemhauser
Integer and Combinatorial Optimization
1988

[8] Christos H. Papadimitriou, Kenneth Steiglitz
Combinatorial Optimization: Algorithms and Complexity
1998

[9] Bertsekas
Non linear programming,
Athenea Sci. 1999

[10] S. Kontogiannis, P. Panagopoulou and P. Spirakis.
Workshop on Internet and Network Economics,
WINE, 2006

[11] R. Lipton, E. Markakis and A. Mehta.
Playing large games using simple strategies,
ACM Electronic Commerce, 2003

[12] I. Althofer.
On sparse approximations to randomized strategies and convex combinations,
Linear Algebra and its Applications, 199, 1994

[13] C. Daskalakis, A. Mehta and C. Papadimitriou
A Note on Approximate Nash Equilibria,
In Proc. of the 2nd W. on Internet and Net Econ. (WINE ’06), vol. 4286 of LNCS, pp. 297-306, Springer, 2006.

[14] X. Chen, X. Deng and S. Teng.
Computing Nash Equilibria: Approximation and Smoothed Complexity,
Electronic Colloquium on Computational Complexity (ECCC), TR06-2023, 2006

[15] D. Koller and N. Megiddo.
Finding mixed strategies with small support in extensive form games.

International Journal of Game Theory, 25:73–92, 1996.
[16] D. Koller, N. Megiddo, and B. von Stengel.
Fast algorithms for finding randomized strategies in game trees.
In Annual ACM Symposium on the Theory of Computing, pages 750–759, 1994.
[17] D. Koller, N. Megiddo, and B. von Stengel.
Efficient computation of equilibria for extensive two-person games.
Games and Economic Behavior, 14(2):247–259, 1996.
[18] H. W. Kuhn.
An algorithm for equilibrium points in bimatrix games.
In Proceedings of the National Academy of Sciences, pages 1657–1662, 1961.
[19] C. E. Lemke.
Bimatrix equilibrium points and mathematical programming.
Management Science, 11:681–689, 1965.
[20] C. E. Lemke and J. T. Howson.
Equilibrium points of bimatrix games.
Journal of the Society for Industrial and Applied Mathematics, 12:413–423, 1964.
[21] O. L. Mangasarian.
Equilibrium points of bimatrix games.
Journal of the Society for Industrial and Applied Mathematics, 12(4):778–780, 1964.
[22] H. Simon.
Models of Bounded Rationality,
Volume 2. MIT Press, Cambridge, Massachusetts, 1982.
[23] A. Rubinstein.
Modeling Bounded Rationality.
MIT Press, Cambridge, Massachusetts, 1998.
[24] C. Daskalakis, A. Mehta and C. Papadimitriou
Progress in approximate nash equilibrium.
In Proc. Of the 8th ACM Conf. on Electr. Commerce (EC'07), 2007.
(to appear).
[25] S. Kontogiannis, P. Panagopoulou and P. Spirakis.
Polynomial algorithms for approximating nash equilibria in bimatrix games.
In Proc. of the 2nd W. on Internet and Net Econ. (WINE '06), vol. 4286 of LNCS, pp. 286-296, Springer, 2006.
[26] H. Boss, J. Byrka and E. Markakis.
New Algorithms for Approximate Nash Equilibria in Bimatrix Games.
In: WINE 2007.
[27] X. Chen, X. Deng and S. Teng.
Computing Nash Equilibria : Approximation and smoothed complexity.
In Proc. of the 47th IEEE Symp. on Foundations of Comp. Sci. (FOCS 06), pp. 603-612, IEEE Press, 2006.
[28] M. Bellare, P. Rogaway.
The complexity of approximating a nonlinear program.
Mathematical Programming, 69:429-441, 1995.
[29] Kernighan, Ritchie
The C Programming Language
2000
[30] Stephen Boyd and Lieven Vandenberghe

Convex Optimization
[31] Vijay Vazirani, Springer Verlag
Approximation Algorithms
1997
[32] Tim Roughgarden, Noam Nisan, Eva Tardos, and Vijay Vazirani,
Algorithmic Game Theory
Cambridge University Press, October 2007.
[33] Σπ. Κοντογιάννης,
Γραμμικός Προγραμματισμός,
Φθινόπωρο 2004.