Συστηματική Αναζήτηση και Ενισχυτική Μάθηση για το Επιτραπέζιο Παιχνίδι Neighbours

Ιωάννης Σκουλάκης

Τμήμα Ηλεκτρονικών Μηχανικών & Μηχανικών Υπολογιστών
Πολυτεχνείο Κρήτης

Εξεταστική Επιτροπή:

Επ. Καθ. Μιχαήλ Γ. Λαγουδάκης (επιβλέπων)
Επ. Καθ. Αντώνιος Δεληγιαννάκης
Επ. Καθ. Νικόλαος Βλάσσης (Τμήμα Μηχανικών Παραγωγής και Διοίκησης)

Χανιά, Φεβρουάριος 2010

Περίληψη

Τα παιχνίδια πάντα αποτελούσαν ένα πολύτιμο τομέα έρευνας και πρακτικής εφαρμογής της Τεχνητής Νοημοσύνης και της Μηχανικής Μάθησης λόγω των νοητικών δεξιοτήτων που απαιτούν. Η παρούσα διπλωματική εργασία επικεντρώνεται σε ένα άγνωστο, αλλά και πολύ απρόβλεπτο, παιχνίδι σκακιέρας, το Neighbours, το οποίο χαρακτηρίζεται από μεγάλο αριθμό επιλογών για κάθε παίκτη σε κάθε κίνηση και από μεγάλο βαθμό εξαρτήσεων μεταξύ των πιονιών πάνω στη σκακιέρα. Στα πλαίσια της εργασίας σχεδιάστηκε και αναπτύχθηκε ένας πράκτορας για το Neighbours, αλλά και ένα γραφικό διαδικτυακό περιβάλλον μέσω του οποίου μπορούν να διεξαχθούν παρτίδες Neighbours με οποιοδήποτε συνδυασμό χρηστών και πρακτόρων. Η στρατηγική του πράκτορά μας για την επιλογή κινήσεων βασίζεται στον αλγόριθμο MiniMax σε συνδυασμό με την τεχνική του κλαδέματος άλφα-βήτα, αλλά και εκείνης του Principal Variation Search (PVS). Το πλήθος των επεκτεινόμενων κόμβων μειώνεται αισθητά με την χρήση ενός Hash Table που χρησιμοποιεί Zobrist Hashing, προσφέροντάς σημαντική βελτίωση σε ταχύτητα. Τέλος, χρησιμοποιώντας μια παραλλαγή του αλγορίθμου ενισχυτικής μάθησης Least-Squares Temporal Difference Learning (LSTD), ο πράκτορας μπορεί να βελτιώνεται ως προς τη συνάρτηση αξιολόγησης καταστάσεων, απλά παίζοντας παρτίδες με τον εαυτό του. Οι τεχνικές βελτιστοποίησης που υλοποιούνται σε συνεργασία με την αποτελεσματική μάθηση δημιουργούν έναν πολύ δυνατό και γρήγορο πράκτορα, ο οποίος μπορεί να εξετάσει κινήσεις μέχρι βάθους 8 σε έναν τυπικό υπολογιστή και πολύ δύσκολα μπορεί να αντιμετωπιστεί από τον άνθρωπο.

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

Το 1949 ο Claude Shannon παρουσίασε δύο βασικές στρατηγικές [1], που θα έδιναν την δυνατότητα σε έναν πράκτορα, να παίξει σκάκι.

Η πρώτη στρατηγική, γνωστή και ως στρατηγική Α του Shannon, αποτελεί μια «brute-force» προσέγγιση στο πρόβλημα. Η επιλογή της κίνησης επιτυγχάνεται εξερευνώντας το σύνολο των διαθέσιμων κινήσεων και των αναμενόμενων συνεπειών τους. Λόγω της έλλειψης υπολογιστικής ισχύος εκείνη την εποχή, ο ίδιος ο Shannon χαρακτήρισε αυτή την στρατηγική (καθώς και τις παραλλαγές της) μη αποδοτική.

Παρατηρώντας ότι η επεξεργασία των κινήσεων είχε μη επιτρεπτό κόστος σε χρόνο, ο Shannon έστρεψε το ενδιαφέρον του στον τρόπο με τον οποίο ο άνθρωπος αντιμετωπίζει το παιχνίδι. Σε αντίθεση με την πρώτη στρατηγική, ο άνθρωπος δεν αναλύει το σύνολο των διαθέσιμων κινήσεων, αλλά έχοντας στο μυαλό του κάποιες συνήθεις καταστάσεις, αποφασίζει να εξετάσει τις κινήσεις που από την εμπειρία του πιστεύει ότι θα του αποφέρουν το μεγαλύτερο κέρδος. Η δεύτερη λοιπόν στρατηγική που παρουσιάζει ο Shannon, προσομοιώνοντας αυτόν ακριβώς τον τρόπο σκέψης, καλείται να επιλέξει ένα μέρος του συνόλου των διαθέσιμων κινήσεων και να το αναλύσει. Με αυτόν τον τρόπο, αποφεύγοντας αρκετές κινήσεις, κερδίζεται χρόνος επεξεργασίας, που μπορεί να χρησιμοποιηθεί στο να εξεταστούν οι (λίγες) επιλεγμένες κινήσεις σε μεγαλύτερο βάθος.

Σήμερα όμως, κατά πρώτον, λόγω της συνεχώς αυξανόμενης υπολογιστικής ισχύος, και κατά δεύτερον, λόγω των βελτιώσεων σε επίπεδο αλγορίθμων, η χρήση της στρατηγικής Α του Shannon αποτελεί τον καθιερωμένο τρόπο επιλογής της βέλτιστης κίνησης από έναν πράκτορα.

1.1 Συνοπτική περιγραφή και συνεισφορά της εργασίας

Στην διπλωματική αυτή εργασία υλοποιούμε έναν πράκτορα για το παιχνίδι σκακιέρας "Neighbours". Πρόκειται για τον πρώτο πράκτορα που υλοποιείται για το εν λόγώ παιχνίδι. Ο πράκτοράς μας χρησιμοποιεί προχωρημένες τεχνικές αναζήτησης επιτυγχάνοντας αρκετά καλές ταχύτητες αναζήτησης και χάρη στην χρήση ενισχυτικής μάθησης μπορεί να βελτιώνεται καθώς παίζει. Παρουσιάζουμε ένα αποδοτικό τρόπο εκπαίδευσης του πράκτορα και προχωράμε στην υλοποίηση ενός γραφικού περιβάλλοντος του παιχνιδιού.

1.2 Διάρθρωση της εργασίας

Στο δεύτερο κεφάλαιο παρουσιάζονται εν συντομία βασικές τεχνικές αναζήτησης, ο αλγόριθμος MiniMax και οι βελτιώσεις του. Επίσης, γίνεται αναφορά στην έννοια της συνάρτησης αξιολόγησης, αλλά και εξηγείται η σημασία της διάταξης των κόμβων-παιδιών και η σπουδαιότητα της ύπαρξης hash table. Τέλος, κλείνουμε με την έννοια της ενισχυτικής μάθησης και τη μέθοδο των χρονικών διαφορών.

Το τρίτο κεφάλαιο περιλαμβάνει μια αναλυτική παρουσίαση του παιχνιδιού Neighbours που μελετήθηκε. Γίνεται αναφορά στις επιδιώξεις και στους στόχους αυτής της διπλωματικής εργασίας και παρατίθεται μια σύντομη επισκόπηση σχετικών εργασιών.

Στο τέταρτο κεφάλαιο μπορεί κάποιος να αντιληφθεί τον τρόπο προσέγγισης που ακολουθήθηκε, καθώς και να μελετήσει μια λεπτομερή αναφορά των χαρακτηριστικών (features) που χρησιμοποιήθηκαν. Επίσης, παρουσιάζεται το μοντέλο μάθησης και η παραλλαγή της μεθόδου LSTD που χρησιμοποιήθηκε.

Το πέμπτο κεφάλαιο περιέχει διάφορα θέματα υλοποίησης, που κρίθηκε αναγκαίο να αναλυθούν σε μεγαλύτερο βάθος. Δίνεται έμφαση στην λειτουργία του hash table και στις επιπτώσεις που έχει η χρήση του, καθώς και στο αποδοτικό σύστημα ελέγχου ισόπαλων καταστάσεων.

Το έκτο κεφάλαιο, παρουσιάζει τα αποτελέσματα της εφαρμογής ενισχυτικής μάθησης στο Neighbours, γίνεται σύγκριση μεταξύ των αλγορίθμων TD και LSTD και αναλύεται το ακριβές μοντέλο μάθησης που χρησιμοποιήθηκε.

Τέλος, στο έβδομο κεφάλαιο παραθέτουμε τα συμπεράσματά μας και προτείνουμε περαιτέρω βελτιώσεις αυτής της εργασίας.

Κεφάλαιο 2

Αναγκαίο γνωστικό υπόβαθρο

2.1 Δέντρο παιχνιδιού και συνάρτηση χρησιμότητας

Υιοθετώντας την Α τύπου στρατηγική του Shannon, πρέπει να βρούμε μία μέθοδο, που θα μας βοηθήσει να αντιληφθούμε τις συνέπειες που θα έχουν οι κινήσεις μας μακροπρόθεσμα. Για να το επιτύχουμε αυτό, προχωράμε στην κατασκευή του δέντρου του παιχνιδιού.

Πρόκειται για ένα δέντρο με ρίζα την αρχική ή κάποια ενδιάμεση κατάσταση του παιχνιδιού μας και παιδιά όλες τις καταστάσεις που προκύπτουν μετά από νόμιμες κινήσεις της κάθε πλευράς εναλλάξ. Το δέντρο επεκτείνεται μέχρι τις καταστάσεις εκείνες στις οποίες ο παίκτης που έχει σειρά να παίξει δεν έχει άλλη νόμιμη κίνηση, σύμφωνα πάντα με τους κανόνες του εκάστοτε παιχνιδιού. Οι καταστάσεις αυτές ονομάζονται τερματικές και αποτελούν τις μοναδικές καταστάσεις, στις οποίες γνωρίζουμε το τελικό αποτέλεσμα του παιχνιδιού.

Στην συνέχεια, εισαγάγουμε την έννοια της συνάρτησης χρησιμότητας. Πρόκειται για μια συνάρτηση, που προσδίδει μία αριθμητική τιμή σε κάθε τερματική κατάσταση, με βάση το αποτέλεσμα του παιχνιδιού. Για παράδειγμα, σε ένα παιχνίδι όπως το σκάκι ή η τρίλιζα, θα μπορούσαμε να αντιστοιχήσουμε την τιμή $+1$ σε περίπτωση νίκης, -1 σε περίπτωση ήττας και 0 σε περίπτωση ισοπαλίας. Ωστόσο, πολλά παιχνίδια διαθέτουν ευρύτερη ποικιλία δυνατών αποτελεσμάτων, με αποτέλεσμα να χρησιμοποιείται μεγαλύτερο εύρος τιμών για την απεικόνιση πιθανής νίκης ή ήττας.

Στόχος μας, λοιπόν, είναι να βρούμε έναν αλγόριθμο, ο οποίος χρησιμοποιώντας το δέντρο παιχνιδιού, να μας βοηθάει να επιλέξουμε κινήσεις, που θα μας οδηγήσουν στις επιθυμητές τερματικές καταστάσεις. Πριν προχωρήσουμε όμως σε αυτόν τον αλγόριθμο, θα αναφερθούμε στην αναζήτηση πρώτα σε βάθος, στον τρόπο δηλαδή, με

τον οποίο διατρέχουμε το δέντρο παιχνιδιού με τον πράκτορά μας.

2.2 Αναζήτηση

2.2.1 Αναζήτηση πρώτα σε βάθος

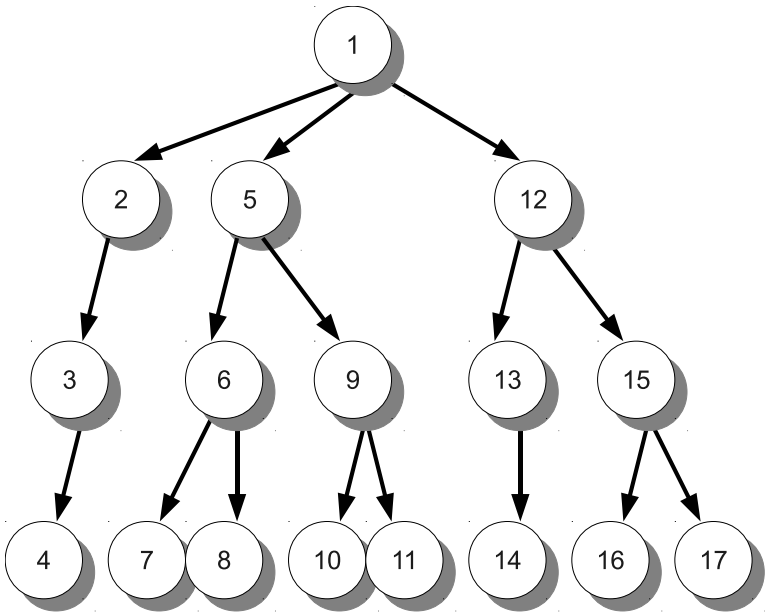
Η αναζήτηση πρώτα σε βάθος (depth-first search) [2], είναι μια μέθοδος, κατά την οποία επεκτείνουμε πάντα το βαθύτερο ανεξερεύνητο κόμβο. Ξεκινώντας από την ρίζα, επεκτείνουμε τον τρέχοντα κόμβο, δημιουργούμε δηλαδή όλα τα παιδιά του, και συνεχίζουμε με τον ίδιο τρόπο αναδρομικά από το πρώτο ανεξερεύνητο παιδί του. Έτσι, διατρέχουμε το δέντρο αναζήτησης, από παιδί σε παιδί, μέχρι να φτάσουμε σε κάποιο φύλλο, και τότε οπισθοχωρούμε στον αμέσως ρηχότερο κόμβο του δέντρου που έχει ανεξερεύνητα παιδιά.

Στο Σχήμα 2.1 φαίνεται ένα παράδειγμα αναζήτησης πρώτα σε βάθος. Οι αριθμοί στο σχήμα δείχνουν την σειρά με την οποία θα επεκταθούν οι κόμβοι. Στο πρώτο βήμα της αναζήτησης, επεκτείνουμε τον κόμβο με τον αριθμό 1, δημιουργώντας τους κόμβους με αριθμούς 2, 5 και 12. Στο δεύτερο βήμα επεκτείνουμε τον 2 και δημιουργούμε τον 3. Ακολούθως στο τρίτο βήμα επεκτείνουμε τον 3 ως βαθύτερο, δημιουργώντας τον 4. Στο τέταρτο βήμα που επεκτείνουμε τον κόμβο με το αριθμό 4, παρατηρούμε ότι πρόκειται για κόμβο-φύλλο συνεπώς ο επόμενος κόμβος που θα επεκταθεί θα είναι ο αμέσως ρηχότερος, δηλαδή ο 5. Συνεχίζουμε ακολουθώντας την ίδια λογική μέχρι τον κόμβο με αριθμό 17.

2.2.2 Αναζήτηση με υπαναχώρηση

Η αναζήτηση με υπαναχώρηση (Backtracking search) [2] αποτελεί μια παραλλαγή της αναζήτησης πρώτα σε βάθος. Κατά την αναζήτηση αυτή, δεν δημιουργούνται όλα τα παιδιά ενός κόμβου όταν αυτός επεκτείνεται, αλλά μόνο ένα. Όταν το υποδέντρο αυτού του παιδιού επεκταθεί πλήρως, τότε δημιουργείται το επόμενο παιδί του κόμβου και η διαδικασία συνεχίζεται αναδρομικά με τον ίδιο τρόπο.

Το σημαντικότερο στοιχείο όμως που εισάγει η αναζήτηση με υπαναχώρηση, είναι η ιδέα ότι μπορούμε να δημιουργήσουμε τον κόμβο-παιδί με απευθείας τροποποίηση του κόμβου-γονέα και εν συνεχεία, όταν αυτό κριθεί αναγκαίο, να γυρίσουμε πίσω στον γονικό κόμβο, με την προϋπόθεση ότι αυτή η λειτουργία θα υποστηρίζεται από τον πράκτορα. Με αυτόν τον τρόπο, αποκομίζουμε τεράστια οφέλη σε χρόνο και χώρο,



Σχήμα 2.1: Σειρά επέκτασης κόμβων σε περίπτωση αναζήτησης πρώτα σε βάθος.

καθώς αποφεύγονται όλες οι αντιγραφές κόμβων σε κάθε επέκταση και κερδίζουμε τον χώρο που θα καταλάμβαναν διαφορετικά.

2.2.3 Αναζήτηση περιορισμένου βάθους

Η αναζήτηση πρώτα σε βάθος έχει και ένα μειονέκτημα. Όταν το βάθος είναι πολύ μεγάλο, αναλώνουμε πάρα πολύ χρόνο αναζήτησης σε ένα συγκεκριμένο υποδέντρο που μπορεί να μην περιέχει καν την τιμή που αναζητούμε, ενώ η λύση μπορεί να βρίσκεται σε κάποιο άλλο πολύ πιο ρηχό υποδέντρο. Επίσης, υπάρχουν και μη φραγμένα δέντρα που καθιστούν την αναζήτηση πρώτα σε βάθος αδύνατη, καθώς δεν τερματίζει ποτέ.

Για να λύσουμε αυτά τα προβλήματα εισάγουμε την έννοια της αναζήτησης περιορισμένου βάθους. Στην ουσία, θέτουμε ένα όριο βάθους, τους κόμβους σ' αυτό το βάθος τούς θεωρούμε φύλλα του δέντρου αναζήτησης, και εκτελούμε κανονικά αναζήτηση πρώτα σε βάθος. Έτσι αποφεύγουμε όλα τα παραπάνω προβλήματα, θυσιάζοντας όμως την πληρότητα της αναζήτησης, καθώς η λύση μπορεί να βρίσκεται σε βάθος μεγαλύτερο του ορίου που έχουμε θέσει.

2.2.4 Αναζήτηση πρώτα σε βάθος με επαναληπτική εκβάθυνση

Η μέθοδος της αναζήτησης πρώτα σε βάθος με επαναληπτική εκβάθυνση, είναι ουσιαστικά η χρήση της αναζήτησης περιορισμένου βάθους, για ολοένα και αυξανόμενο όριο βάθους. Σύμφωνα με αυτήν την μέθοδο, κάνουμε αρχικά μια αναζήτηση περιορισμένου βάθους με όριο βάθους 1, στη συνέχεια, αν η αναζήτηση αποτύχει, ξανακάνουμε αναζήτηση περιορισμένου βάθους με όριο βάθους 2, μετέπειτα με όριο βάθους 3, κ.ο.κ.

Μπορεί όλη αυτή η διαδικασία να ακούγεται σπάταλη. Στην πράξη όμως, η επιβάρυνση είναι μικρή, καθώς οι περισσότεροι κόμβοι βρίσκονται στο μεγαλύτερο, άρα και τελευταίο βάθος. Τα οφέλη όμως που αποκτούμε, όπως θα δούμε, είναι πολλαπλάσια, και δεν είναι άλλα από τις πληροφορίες που μας παρέχουν οι αναζητήσεις μικρότερου βάθους.

2.3 Αλγόριθμος MiniMax και βελτιώσεις του

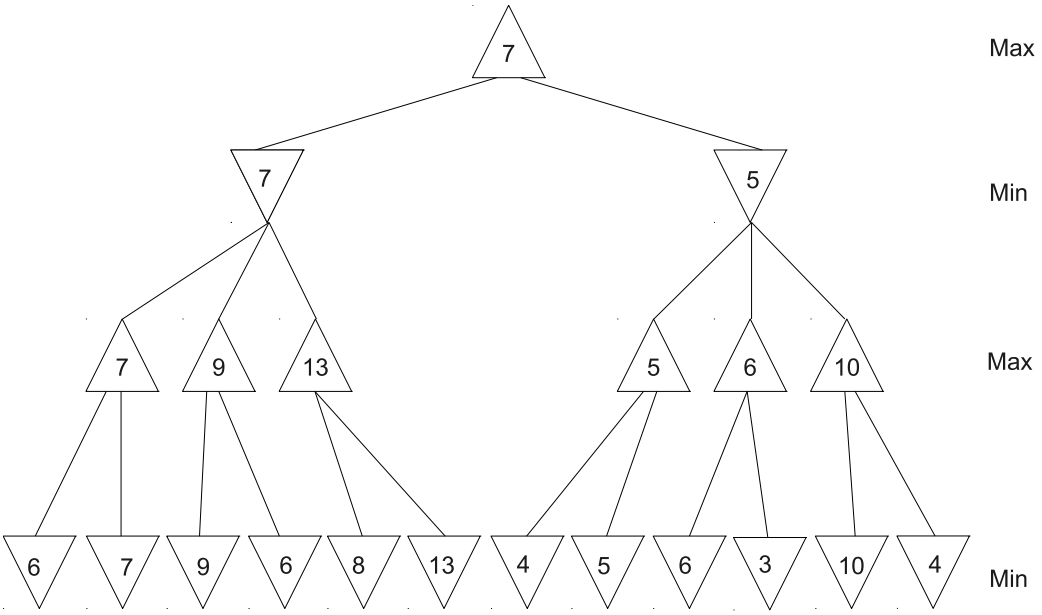
2.3.1 Αλγόριθμος MiniMax

Παρατηρώντας το δέντρο του παιχνιδιού, βλέπουμε μονοπάτια που μας οδηγούν σε επιθυμητές τερματικές καταστάσεις. Ιδανικά θα επιθυμούσαμε, να βρεθούμε στην τερματική κατάσταση με το μεγαλύτερο κέρδος. Όπως θα διαπιστώσουμε όμως, ο στόχος μας, να βρούμε το μονοπάτι που θα μας οδηγήσει σε αυτήν την τερματική κατάσταση, μετατρέπεται σε ένα κυνήγι τιμών.

Σε αυτό το παιχνίδι δεν παίζουμε μόνοι μας. Έχουμε απέναντι μας έναν αντίπαλο, άνθρωπο ή πράκτορα, που έχει τους ίδιους στόχους με εμάς. Έτσι πρέπει να βρούμε έναν αλγόριθμο που, λαμβάνοντας υπόψη του την ύπαρξη και τις επιθυμίες του αντιπάλου, να μας οδηγεί στην επιλογή των «κατάλληλων» κάθε φορά κινήσεων για να οδηγηθούμε στην κατάσταση με τη μεγαλύτερη δυνατή τιμή κέρδους.

Για την επιλογή αυτών των «κατάλληλων κινήσεων» χρησιμοποιούνται αλγόριθμοι που βασίζονται στον αλγόριθμο MiniMax [2]. Ο αλγόριθμος αυτός προσομοιώνει την επιθυμία μας, να μεγιστοποιήσουμε το κέρδος μας, αλλά και την επιθυμία του αντιπάλου, να το ελαχιστοποιήσει. Γι' αυτό τον λόγο από εδώ και στο εξής θα ονομάζουμε τον παίκτη μας Max και τον αντίπαλό μας Min. Αναλυτικότερα, ο MiniMax αποτελείται από δύο κύριες συναρτήσεις, μία για κάθε παίκτη. Η πρώτη, που εκτελείται στους κόμβους που πρέπει να παίξουμε εμείς, επιλέγει την διάδοχη κατάσταση με το μεγαλύτερο δυνατό κέρδος, ενώ η δεύτερη, που εκτελείται στους κόμβους που παίζει ο αντίπαλος, επιλέγει εκείνη με το μικρότερο. Για να κατανοήσουμε καλύτερα αυτόν τον αλγόριθμο

μπορούμε να δούμε το δέντρο παιχνιδιού που φαίνεται στο Σχήμα 2.2.



Σχήμα 2.2: Παράδειγμα εκτέλεσης αλγορίθμου MiniMax.

Το δέντρο μας έχει συνολικό βάθος δύο κινήσεων (η κίνηση θεωρείται ολοκληρωμένη όταν και οι δύο παίκτες έχουν κινηθεί). Η κίνηση του ενός μόνο παίκτη, η μισή δηλαδή κανονική κίνηση, ονομάζεται στρώση (ply). Συνεπώς, αυτό το δέντρο έχει συνολικά τέσσερις στρώσεις. Τα σημεία στα οποία καλούμαστε να παίξουμε, έχουν σχήμα $\triangle$, ενώ εκείνα στα οποία ο αντίπαλος μας καλείτε να επιλέξει την κίνησή του, έχουν σχήμα $\bigtriangledown$. Όπως προείπαμε, στις στρώσεις που παίζουμε εμείς, προσπαθούμε να μεγιστοποιήσουμε το κέρδος μας, συνεπώς επιλέγουμε την μέγιστη δυνατή τιμή από τους διάδοχους κόμβους. Έτσι λοιπόν για παράδειγμα, μεταξύ των δύο κινήσεων που έχουν τιμές 9 και 6, επιλέγουμε το 9, και το αναγράφουμε στον πατρικό κόμβο. Όταν τώρα παίζει ο αντίπαλος, συμβαίνει ακριβώς το αντίθετο, γι' αυτό και ανάμεσα στις τιμές 7, 9 και 13, επιλέγεται η μικρότερη, δηλαδή το 7.

Ακολουθώντας αυτή τη διαδικασία σε κάθε κόμβο και δημιουργώντας το δέντρο με πρώτα σε βάθος αναζήτηση προσδίδουμε τελικά στην αρχική μας κατάσταση μία τιμή. Η τιμή αυτή ονομάζεται τιμή MiniMax και για να βρούμε το μονοπάτι που μας την προσφέρει, απλά αναζητούμε ποιο ή ποια από τα παιδιά-καταστάσεις έχουν αυτήν την τιμή. Στο παράδειγμά μας, η τιμή MiniMax είναι η τιμή 7. Αυτό σημαίνει ότι, ανεξαρτήτως κινήσεων αντιπάλου, όταν θα φτάσουμε σε τερματικό κόμβο, θα έχουμε τιμή κέρδους ίση ή μεγαλύτερη του 7. Ο αλγόριθμος MiniMax σε ψευδοκώδικα φαίνεται παρακάτω ως Algorithm 1. Η συνάρτηση Utility επιστρέφει την τιμή χρησιμότητας της

κατάστασης, ενώ η συνάρτηση Successors δημιουργεί ένα σύνολο με όλες τις διάδοχες καταστάσεις της δοθείσας κατάστασης.

Algorithm 1 Ο αλγόριθμος MiniMax

Function MiniMaxDecision (κατάσταση)
inputs: Κατάσταση, τρέχουσα κατάσταση παιχνιδιού
returns: Μια ενέργεια
  $u \leftarrow$ MaxValue (κατάσταση)
  return Ενέργεια $\in$ Successors (κατάσταση) που έχει τιμή u

Function MaxValue (κατάσταση)
inputs: Μια κατάσταση
returns: Μια τιμή χρησιμότητας
  if TerminalTest (κατάσταση) **then**
    return Utility (κατάσταση)
  end if
  $u \leftarrow -\infty$
  for all $s \in$ Successors (κατάσταση) **do**
    $u \leftarrow \max(u,$ MinValue (κατάσταση))
  end for
  return u

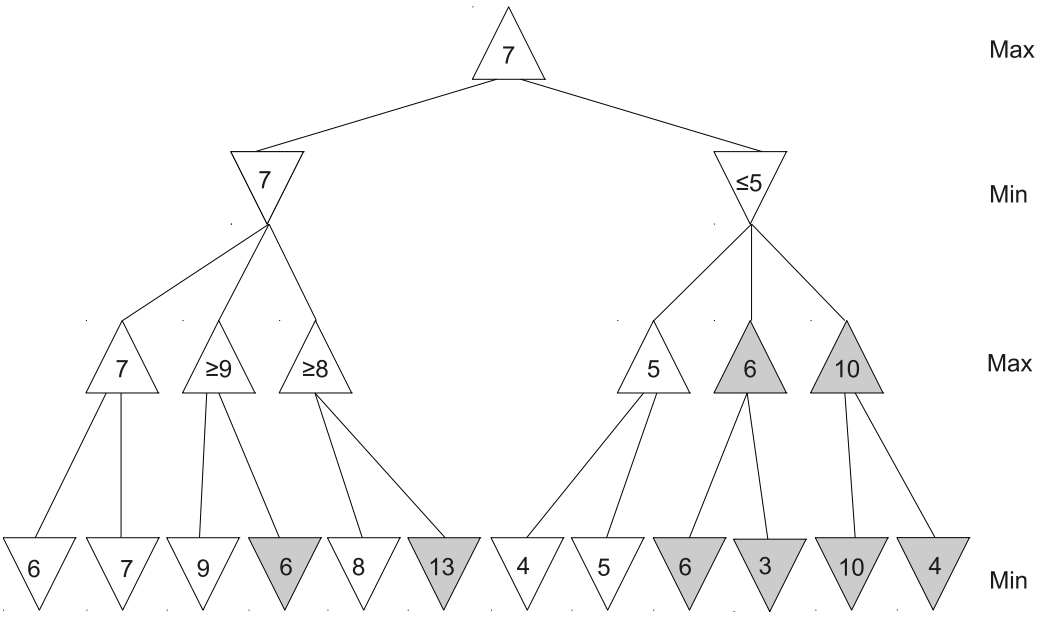
Function MinValue (κατάσταση)
inputs: Μια κατάσταση
returns: Μια τιμή χρησιμότητας
  if TerminalTest (κατάσταση) **then**
    return Utility (κατάσταση)
  end if
  $u \leftarrow +\infty$
  for all $s \in$ Successors (κατάσταση) **do**
    $u \leftarrow \min(u,$ MaxValue (κατάσταση))
  end for
  return u

2.3.2 Κλάδεμα άλφα-βήτα

Παρατηρώντας το παράδειγμα που δώσαμε για τον αλγόριθμο MiniMax, μπορεί κανείς να αντιληφθεί ότι υπάρχουν υποδέντρα που δεν επηρεάζουν την MiniMax τιμή, και κατ' επέκταση ούτε την απόφαση. Για παράδειγμα, το να συνεχίσουμε να επεκτείνουμε ένα υποδέντρο που μας δίνει τιμή το πολύ $+3$, ενώ έχουμε εξασφαλίσει το $+4$, είναι εντελώς άσκοπο. Το άλφα-βήτα κλάδεμα είναι μια βελτίωση του αλγορίθμου MiniMax, που του δίνει την δυνατότητα να «κλαδεύει» αυτά τα υποδέντρα.

Αυτό το οποίο κάνουμε, είναι να αποθηκεύουμε μαζί με την κατάσταση δύο τιμές, την τιμή άλφα και την τιμή βήτα. Η τιμή άλφα είναι η καλύτερη (μεγαλύτερη) τιμή που έχουμε βρει σε οποιοδήποτε σημείο κατά μήκος της διαδρομής για τον παίκτη Max, ενώ η τιμή βήτα είναι η καλύτερη (μικρότερη) τιμή που έχουμε βρει σε οποιοδήποτε σημείο κατά μήκος της διαδρομής για τον παίκτη Min. Με αυτόν τον τρόπο, μπορούμε να ελέγξουμε αν ακολουθούμε κάποια «άσκοπη» διαδρομή και να την εγκαταλείψουμε.

Στο παράδειγμα που φέραμε για τον MiniMax τέσσερα υποδέντρα θα «κλαδευτούν» από την προσθήκη του άλφα-βήτα κλαδέματος [2]. Θα γλιτώναμε δηλαδή την επέκταση των γκρίζων κόμβων, όπως φαίνεται στο Σχήμα 2.3.



Σχήμα 2.3: Παράδειγμα εκτέλεσης αλγορίθμου MiniMax με άλφα-βήτα κλάδεμα.

Ο αριθμός των υποδέντρων που κλαδεύονται, ή αν προτιμάτε ο βαθμός επιτυχίας του άλφα-βήτα κλαδέματος, σχετίζεται άμεσα με τη σειρά με την οποία θα επεκτείνουμε τους κόμβους. Βέλτιστα, θα επιθυμούσαμε να επεκτείνουμε, στις στρώσεις του

Max, πρώτα τις καταστάσεις-παιδιά που θα οδηγήσουν στις μεγαλύτερες τιμές και στις στρώσεις που παίζει ο Min το ακριβώς αντίθετο. Θα θέλαμε δηλαδή η σειρά επέκτασης να μας δίνει πάντα την καλύτερη κίνηση για τον εκάστοτε παίκτη πρώτη. Με αυτόν τον τρόπο, κλαδεύονται οι περισσότεροι δυνατοί κόμβοι και χρειάζεται να εξετάσουμε περίπου μόνο $O(b^{\frac{m}{2}})$ κόμβους, όπου m είναι ο αριθμός των στρώσεων και b είναι ο παράγοντας διακλάδωσης, ο μέσος δηλαδή αριθμός των παιδιών κάθε κόμβου που βρίσκεται στο δέντρο. Η τιμή αυτή είναι πολύ μικρότερη από τους $O(b^m)$ κόμβους που απαιτεί ο MiniMax. Αυτό στην ουσία σημαίνει ότι ο παράγοντας διακλάδωσης γίνεται $\sqrt{b}$ από b, κάτι το οποίο έχει τεράστιο αντίκτυπο στον αριθμό των κόμβων που πρέπει να διατρέξουμε, επιτρέποντάς μας να διατρέχουμε δύο φορές βαθύτερα δέντρα παιχνιδιού από τον MiniMax στον ίδιο χρόνο.

Με την προσθήκη του άλφα-βήτα κλαδέματος, ο αλγόριθμος αναζήτησής μας, διαμορφώνεται όπως φαίνεται παρακάτω (Algorithm 2).

Algorithm 2 Ο αλγόριθμος άλφα-βήτα

Function AlphaBetaSearch (κατάσταση)

inputs: Κατάσταση, τρέχουσα κατάσταση παιχνιδιού

returns: Μια ενέργεια

$u \leftarrow$ MaxValue (κατάσταση, $-\infty$, $+\infty$)

return Ενέργεια $\in$ Successors (κατάσταση) που έχει τιμή u

Function MaxValue (κατάσταση, α, β)

inputs: Μια κατάσταση

α, τιμή καλύτερης εναλλακτικής επιλογής του Max πάνω στην διαδρομή προς την κατάσταση

β,τιμή καλύτερης εναλλακτικής επιλογής του Min πάνω στην διαδρομή προς την κατάσταση

returns: Μια τιμή χρησιμότητας

if TerminalTest (κατάσταση) **then**

return Utility (κατάσταση)

end if

$u \leftarrow -\infty$

for all $s \in$ Successors (κατάσταση) **do**

$u \leftarrow \max(u,$ MinValue (κατάσταση, α, $\beta))$

if $u \geq \beta$ **then**

return u

end if

$\alpha \leftarrow \max(\alpha, u)$

end for

return u

Function MinValue (κατάσταση, α, β)
inputs: Μια κατάσταση
α, τιμή καλύτερης εναλλακτικής επιλογής του Max πάνω στην διαδρομή προς την κατάσταση
β,τιμή καλύτερης εναλλακτικής επιλογής του Min πάνω στην διαδρομή προς την κατάσταση
returns: Μια τιμή χρησιμότητας

```
if TerminalTest (κατάσταση) then
  return Utility (κατάσταση)
end if
u ← +∞
for all s ∈ Successors (κατάσταση) do
  u ← min(u, MaxValue (κατάσταση, α, β))
  if u ≤ α then
    return u
  end if
  β ← min(β, u)
end for
return u
```

2.3.3 Principal Variation Search (PVS)

Ο PVS [3] αποτελεί μία έξυπνη παραμετροποίηση του MiniMax με άλφα-βήτα κλάδεμα, που μειώνει περαιτέρω το πλήθος των κόμβων που εξερευνούμε, εκμεταλλευόμενος την καλή διάταξη των παιδιών πριν την επέκτασή τους. Ουσιαστικά υποθέτουμε ότι η πρώτη κίνηση που θα επεκτείνουμε θα είναι πιθανότατα και η καλύτερη για τον παίκτη που έχει σειρά. Αν αυτή η υπόθεση ισχύει, θα έχουμε μεγάλο όφελος, αν όχι, απλά επαναλαμβάνουμε χωρίς καμία υπόθεση.

Έστω λοιπόν ότι βρισκόμαστε σε κόμβο που πρέπει να επιλέξουμε δική μας κίνηση, παίζει δηλαδή ο Max. Επιλέγουμε κανονικά τον πρώτο διάδοχο κόμβο και εκτελούμε κανονικά με όρια άλφα και βήτα την MinValue, όπως θα κάναμε και στην περίπτωση του MiniMax με άλφα-βήτα κλάδεμα. Έτσι λαμβάνουμε μια τιμή η οποία, εξαιτίας του ότι βρισκόμαστε στο πρώτο παιδί, γίνεται το νέο μας άλφα όριο. Όμως, όπως είπαμε πριν, θεωρούμε ότι αυτή η τιμή είναι και η πραγματική τιμή και συνεπώς η καλύτερη σε σύγκριση με όλα τα υπόλοιπα παιδιά του ίδιου κόμβου. Τώρα εκτελούμε την MinValue για τα υπόλοιπα παιδιά με όρια άλφα και άλφα+ϵ, όπου το ϵ είναι μια πολύ μικρή θετική

σταθερά (σε περίπτωση ακέραιων τιμών το ϵ είναι ίσο με την μονάδα, ενώ σε περίπτωση πραγματικών τιμών το ε μπορεί να είναι πολύ μικρότερο). Λόγω της υπόθεσής μας, περιμένουμε να έχουμε fail-low, δηλαδή τιμή επιστροφής μικρότερη του άλφα. Αν γίνει αυτό, τότε απλά συνεχίζουμε, έχοντας γλιτώσει την επέκταση αρκετών κόμβων, καθώς εκτελέσαμε την MinValue με πάρα πολύ μικρό παράθυρο άλφα-βήτα. Αν δεν έχουμε fail-low, τότε η υπόθεση ήταν λανθασμένη και εκτελούμε ξανά την MinValue με όρια το value-ϵ και βήτα, όπου value η τιμή που μας επέστρεψε η MinValue όταν δοκιμάζαμε να επαληθεύσουμε την υπόθεσή μας.

Ακολουθώντας το ίδιο σκεπτικό και στην περίπτωση που παίζει ο Min, εκτελούμε αντίστοιχα την MaxValue με όρια βήτα-ϵ και βήτα προσδοκώντας σε κάποια τιμή που θα δημιουργήσει fail-high. Συνεπώς, ο αλγόριθμος του MiniMax με άλφα-βήτα κλάδεμα για να συμπεριλάβει το PVS τροποποιείται όπως φαίνεται παρακάτω (Algorithm 3).

Algorithm 3 Principal Variation Search

Function AlphaBetaSearch (κατάσταση)

inputs: Κατάσταση, τρέχουσα κατάσταση παιχνιδιού

returns: Μια ενέργεια

$u \leftarrow$ MaxValue (κατάσταση, $-\infty$, $+\infty$)

return Ενέργεια $\in$ Successors (κατάσταση) που έχει τιμή u

Function MaxValue (κατάσταση, α, β)

inputs: Μια κατάσταση

α, τιμή καλύτερης εναλλακτικής επιλογής του Max πάνω στην διαδρομή προς την κατάσταση

β,τιμή καλύτερης εναλλακτικής επιλογής του Min πάνω στην διαδρομή προς την κατάσταση

returns: Μια τιμή χρησιμότητας

```
if TerminalTest (κατάσταση) then
  return Utility (κατάσταση)
end if
u ← −∞
for all s ∈ Successors (κατάσταση) do
  if s πρώτο παιδί then
    u ← max(u, MinValue (κατάσταση, α, β))
  else
    value ← max(u, MinValue (κατάσταση, α, α + ε))
    if value > α then
      value ← max(u, MinValue (κατάσταση, value − ε, β))
    end if
    u ← value
  end if
  if u ≥ β then
    return u
  end if
  α ← max(α, u)
end for
return u
```

Function MinValue (κατάσταση, α, β)
inputs: Μια κατάσταση
α, τιμή καλύτερης εναλλακτικής επιλογής του Max πάνω στην διαδρομή προς την κατάσταση
β,τιμή καλύτερης εναλλακτικής επιλογής του Min πάνω στην διαδρομή προς την κατάσταση
returns: Μια τιμή χρησιμότητας

```
if TerminalTest (κατάσταση) then
  return Utility (κατάσταση)
end if
u ← +∞
for all s ∈ Successors (κατάσταση) do
  if s πρώτο παιδί then
    u ← min(u, MaxValue (κατάσταση, α, β))
  else
    value ← min(u, MaxValue (κατάσταση, β − ϵ, β))
    if value < β then
      value ← min(u, MaxValue (κατάσταση, α, value + ϵ))
    end if
    u ← value
  end if
  if u ≤ α then
    return u
  end if
  β ← min(β, u)
end for
return u
```

2.4 Συνάρτηση αξιολόγησης

Βασιζόμενοι λοιπόν στους παραπάνω αλγορίθμους, λύνουμε το πρόβλημα της αναζήτησης των «σωστών» κινήσεων. Συνεπώς, όταν έχουμε τη δυνατότητα να διατρέξουμε

ολόκληρο το δέντρο ενός παιχνιδιού, μπορούμε εύκολα να βρούμε τις κινήσεις που θα μας οδηγήσουν στο στόχο μας. Όταν όμως αυτό το δέντρο είναι πάρα πολύ μεγάλο ή λόγω κάποιου περιορισμού δεν μπορούμε να το διατρέξουμε ολόκληρο, τότε δεν μπορούμε να ξέρουμε ποιες κινήσεις θα μας οδηγήσουν σε επιθυμητό αποτέλεσμα. Η λύση είναι να εξερευνήσουμε το δέντρο παιχνιδιού μέχρι κάποιο συγκεκριμένο βάθος και με βάση τα αποτελέσματα αυτής της μερικής αναζήτησης να επιλέξουμε την κίνησή μας. Πώς όμως μπορούμε να αποφανθούμε για το ποια είναι η κατάλληλη κίνηση, όταν το δέντρο μας μπορεί να μην περιέχει κανένα τερματικό κόμβο;

Το πρόβλημα αυτό έρχεται να λύσει η συνάρτηση αξιολόγησης. Η συνάρτηση αυτή έχει ως στόχο να αξιολογεί τις καταστάσεις (ενδιάμεσες ή τερματικές), κατά τέτοιο τρόπο ώστε, όσο καλύτερη είναι μια κατάσταση για εμάς τόσο μεγαλύτερη τιμή να αποκτά. Έτσι, ο αλγόριθμος αναζήτησης θα επιλέξει το μονοπάτι που θα μας οδηγήσει στην μεγαλύτερη δυνατή τιμή αξιολόγησης, χωρίς πάντα αυτό να σημαίνει ότι επρόκειτο για τη βέλτιστη τιμή του παιχνιδιού. Συνήθως, για συνάρτηση αξιολόγησης χρησιμοποιείται μια σταθμισμένη γραμμική συνάρτηση:

$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + ... + w_n f_n(s) = \sum_{i=1}^{n} w_i f_i(s)$$

Τα $f_i(s)$ αποτελούν τις τιμές των χαρακτηριστικών (features) που εντοπίζουμε σε μία κατάσταση s, ενώ τα w_i τα βάρη που αντικατοπτρίζουν τη σχετική σπουδαιότητα αυτών των χαρακτηριστικών. Όπως τονίστηκε προηγουμένως, όταν δεν έχουμε ολόκληρο το δέντρο του παιχνιδιού η απόφασή μας βασίζεται σε αυτήν την συνάρτηση, καθιστώντας την ένα πολύ σημαντικό μέρος του πράκτορά μας, επηρεάζοντας σε μεγάλο βαθμό τις ικανότητές του. Τα στοιχεία που δημιουργούν μία δυνατή σταθμισμένη γραμμική συνάρτηση είναι η επιλογή των χαρακτηριστικών, αλλά και η σωστή στάθμισή τους.

2.5 Διάταξη διάδοχων κόμβων

2.5.1 Σημασία διάταξης παιδιών

Όπως έχει γίνει αντιληπτό, η διάταξη των παιδιών παίζει πάρα πολύ μεγάλο ρόλο στον αριθμό των κόμβων που θα εξερευνήσουμε και κατ' επέκταση στο χρόνο που θα σπαταλήσουμε για να επιλέξουμε την κίνησή μας. Όπως προείπαμε, αν στους κόμβους που εξετάζουμε δικές μας κινήσεις και προσπαθούμε να μεγιστοποιήσουμε το κέρδος μας, προκύψει υποδέντρο με τιμή μεγαλύτερη αυτής του βήτα, τότε η αναζήτηση σε αυτούς

τους κόμβους σταματά (βήτα-cutoff ή αλλιώς fail-high). Αντίστοιχα αν σε κόμβους που εξετάζονται κινήσεις του αντιπάλου, προκύψει υποδέντρο με τιμή μικρότερη αυτής του άλφα, τότε και πάλι η αναζήτηση σταματά (άλφα-cutoff ή αλλιώς fail-low). Κατά συνέπεια, όσο γρηγορότερα εξετάσουμε κόμβους που μας οδηγούν σε μελλοντικά cutoff, τόσα περισσότερα υποδέντρα θα κλαδευτούν και τόσο περισσότερο χρόνο θα γλιτώσουμε από τις άσκοπες εξερευνήσεις αυτών των υποδέντρων που δεν επηρεάζουν την τελική μας απόφαση.

Λόγω αυτής της ιδιαίτερης σχέσης, γίνεται αντιληπτό ότι αξίζει να διαθέσουμε κάποιο χρόνο ώστε να βελτιώσουμε τη σειρά επέκτασης των εκάστοτε παιδιών-κόμβων, καθώς τα οφέλη, όπως θα διαπιστώσουμε, είναι πολλαπλάσια. Προφανώς, δεν μπορούμε να διατάσσουμε πάντα τους διάδοχους κόμβους βέλτιστα, καθώς αν ξέραμε ποιος είναι καλύτερος, δεν θα μπαίναμε στον κόπο της αναζήτησης.

2.5.2 Hash Table

Από τη στιγμή που χρησιμοποιούμε τόσους πολλούς πόρους για να εντοπίζουμε Mini-Max τιμές, γιατί να μην τις αποθηκεύουμε για να τις εκμεταλλευτούμε στο μέλλον; Αυτό ακριβώς κάνει το hash table. Είναι στην ουσία ένας μεγάλος πίνακας, στον οποίο αποθηκεύουμε χρήσιμες πληροφορίες που παράγονται από τις αναζητήσεις μας.

Πληροφορίες, όπως για παράδειγμα, η τιμή ενός κόμβου ή τα όρια με τα οποία έφτασε η αναζήτηση σε αυτόν. Για έναν πράκτορα, που βασίζεται σε αλγορίθμους που εκμεταλλεύονται το άλφα-βήτα κλάδεμα, το hash table είναι ιδιαίτερα σημαντικό. Αυτό θα γίνει αντιληπτό στη συνέχεια, όταν θα δείξουμε, πως εκμεταλλευόμενοι τις αποθηκευμένες τιμές θα μπορούμε να διατάσσουμε με μεγάλη επιτυχία τους διάδοχους κόμβους που θα προκύπτουν.

Το μεγαλύτερο πρόβλημα που συναντάμε όμως στην υλοποίηση της ιδέας του hash table, είναι πως θα μπορέσουμε να αντιστοιχήσουμε τις θέσεις του πίνακα σε καταστάσεις. Αυτό που χρειαζόμαστε είναι ένα πολύ καλό hash function που να αντιστοιχεί έναν αριθμό σε κάθε κατάσταση, όσο γίνεται πιο μοναδικά και όσο γίνεται πιο γρήγορα, καθώς αυτή η διαδικασία αναμένεται να γίνετε εκατοντάδες χιλιάδες φορές, αν όχι εκατομμύρια φορές, σε κάθε αναζήτηση για την βέλτιστη κίνηση.

2.5.3 Zobrist hashing

Τη λύση στο παραπάνω πρόβλημα έρχεται να δώσει το Zobrist hashing [6]. Πρόκειται για ένα hash function εμπνευσμένο ειδικά για την αντιστοίχιση καταστάσεων σκακιέ-

ρας σε αριθμούς.

Αρχικά, δημιουργούμε έναν δισδιάστατο πίνακα με τυχαίους αριθμούς, όπου η μία διάσταση είναι ίση με των αριθμό των τετραγώνων της σκακιέρας, και η δεύτερη ίση με το πλήθος των διαφορετικών κομματιών που υπάρχουν στο παιχνίδι, δικά μας ή του αντιπάλου. Για παράδειγμα, στο σκάκι θα έχουμε έναν πίνακα με την πρώτη διάσταση ίση με 64 και την δεύτερη ίση με 6 (κομμάτια) $\times 2$ (παίκτες) $= 12$. Άρα, ο πίνακάς μας θα έχει συνολικά $12 \times 64 = 768$ τυχαίους αριθμούς. Στην συνέχεια για κάθε σκακιέρα, που θα θέλουμε να βρούμε τον αριθμό στον οποίο αντιστοιχεί σύμφωνα με το Zobrist hashing, θα πρέπει να διατρέξουμε όλη την σκακιέρα, να επιλέξουμε τους αντίστοιχους τυχαίους αριθμούς με βάση τα κομμάτια και τα τετράγωνα στα οποία βρίσκονται και χρησιμοποιώντας την πράξη XOR να δημιουργήσουμε έναν νέο αριθμό, από όλους τους προηγούμενους, που σε αυτήν την περίπτωση αντιπροσωπεύει τη συνολική κατάσταση της σκακιέρας.

Αν λοιπόν στο σκάκι θέλουμε να βρούμε τον Zobrist αριθμό μιας σκακιέρας, διατρέχουμε την σκακιέρα μέχρι να βρούμε κατειλημμένο τετράγωνο. Έστω ότι αυτό το τετράγωνο είναι το 24 και το κομμάτι που βρίσκεται εκεί είναι ένα αντίπαλο πιονάκι. Θα πάμε στον πίνακα Zobrist και θα πάρουμε τον τυχαίο αριθμό που βρίσκεται στην θέση ZobristTable$[24][7]$, όπου το 7 αντιπροσωπεύει το αντίπαλο πιόνι. Από αυτόν τον αριθμό και όλους τους υπόλοιπους που θα επιλέξουμε με τον ίδιο τρόπο διατρέχοντας όλη την σκακιέρα και με την χρήση της πράξης XOR θα οδηγηθούμε σε έναν αριθμό Zobrist που θα αντιπροσωπεύει την σκακιέρα.

Αυτή η μέθοδος έχει δύο πολύ σημαντικά πλεονεκτήματα. Πρώτον αν αλλάξει έστω και ένα τετράγωνο της σκακιέρας, καταλήγουμε σε έναν άλλο αριθμό που διαφέρει κατά πολύ από τον προηγούμενο, και κατά συνέπεια μειώνεται πολύ η πιθανότητα να έχουμε hash error. Και δεύτερον, η κίνηση ενός πιονιού σε μια σκακιέρα, ισοδυναμεί με τρία το πολύ (δύο αν δεν έχουμε «αιχμαλώτιση») XOR για να μεταβούμε, από το Zobrist αριθμό της σκακιέρας αυτής, στο Zobrist αριθμό της παραγόμενης.

2.6 Ενισχυτική μάθηση

Όπως σημειώσαμε νωρίτερα, για να δημιουργήσουμε μία δυνατή συνάρτηση αξιολόγησης, πρέπει να επιλέξουμε σημαντικά για το παιχνίδι χαρακτηριστικά και να τους αποδώσουμε τα κατάλληλα βάρη. Για τη σωστή επιλογή χαρακτηριστικών δεν υπάρχει κάποια εγγυημένα βέλτιστη μέθοδος. Το μόνο το οποίο μπορούμε να κάνουμε είναι να επιλέξουμε κάποια σετ χαρακτηριστικών και όταν βρούμε κατάλληλα για αυτά βάρη

να τα συγκρίνουμε μεταξύ τους, με γνώμονα την απόδοσή τους στο παιχνίδι. Επικεντρωνόμαστε λοιπόν στον τρόπο με τον οποίο θα μπορέσουμε να βρούμε κατάλληλες τιμές για τα βάρη.

Η μέθοδος μάθησης που ταιριάζει στο πρόβλημά μας είναι η ενισχυτική μορφή μάθησης [2, 4, 5]. Σε αυτήν την μέθοδο, αφήνουμε τον πράκτορά μας να παίξει το παιχνίδι με κάποιες τιμές βαρών, χωρίς κάποια άλλη προηγούμενη γνώση. Όταν λοιπόν οδηγηθούμε σε κάποιο τερματικό κόμβο, τότε καλούμε τον πράκτορά μας να μάθει από τις συνέπειες των επιλογών του, βαθμολογώντας τον ανάλογα με το τελικό αποτέλεσμα. Η θετική βαθμολογία ενισχύει τις επιλογές του, ενώ η αρνητική τις αποθαρρύνει.

2.6.1 Μάθηση χρονικών διαφορών (TD)

Μια από τις πολλές μορφές ενισχυτικής μάθησης είναι η μάθηση χρονικών διαφορών [4]. Σε αυτή τη μορφή μάθησης, προσπαθούμε να μάθουμε τις σωστές τιμές που πρέπει να έχουν τα βάρη μας, χρησιμοποιώντας τις μεταβάσεις από κατάσταση σε κατάσταση. Στόχος, σε κάθε βήμα, είναι να εξαλείψουμε την διαφορά στις τιμές που δίνει η συνάρτηση αξιολόγησης ανάμεσα σε δύο διαδοχικές καταστάσεις (s και s'), όταν ο παίκτης μας επιλέγει την κίνηση που φαίνεται καλύτερη βάσει της τρέχουσας γνώσης του.

Σε αυτό το σημείο, πρέπει να τονιστεί ότι η επιλογή του s' μπορεί να γίνει με δύο μεθόδους. Στην πρώτη, επιλέγουμε το s' να είναι η κατάσταση, στην οποία θα οδηγηθούμε, αν ο αντίπαλός μας, απαντήσει στην κίνηση που επιλέξαμε από την αναζήτησή μας, με την κίνηση που προβλέψαμε ότι θα απαντήσει. Ενώ στην δεύτερη μέθοδο, επιλέγουμε το s' να είναι η κατάσταση στην οποία οδηγούμαστε, μετά την πραγματική απάντηση του αντιπάλου, στην κίνησή μας. Ουσιαστικά, με την πρώτη μέθοδο, μαθαίνουμε παίζοντας απέναντι στον εαυτό μας, επικεντρώνοντας την προσοχή μας σε βέλτιστους ως προς την γνώση μας αντιπάλους. Σ' αυτήν την περίπτωση οι πραγματικές κινήσεις του αντιπάλου είναι αδιάφορες και μπορούν να επιλεγούν ώστε να εξερευνήσουμε όσο το δυνατόν μεγαλύτερο τμήμα του δέντρου του παιχνιδιού. Με την χρήση της δεύτερης μεθόδου, μαθαίνουμε από τις πραγματικές κινήσεις του αντιπάλου και έτσι μαθαίνουμε να παίζουμε εναντίον του συγκεκριμένου αντιπάλου. Προφανώς, όταν δεν υπάρχει κάποιος άλλος συγκεκριμένος πράκτορας, που να θέλουμε να ανταγωνιστούμε, όπως δηλαδή στην περίπτωσή μας, χρησιμοποιείται η πρώτη μέθοδος.

Για την ενημέρωση των βαρών, χρησιμοποιείται σε κάθε κίνηση ο ακόλουθος κανόνας:

$$w_i \leftarrow w_i + \alpha f_i(s)(r + \gamma Eval(s') - Eval(s))$$

Όπου το α είναι ο ρυθμός μάθησης και γ ο παράγοντας προεξόφλησης. Ως ρυθμός μάθησης συνήθως επιλέγεται ένας πολύ μικρός αριθμός, που καθορίζει το βάρος που δίνουμε στις μεταβολές που προτείνει το κάθε βήμα της μάθησης. Ο παράγοντας προεξόφλησης από την άλλη, είναι ένας αριθμός ελαφρά μικρότερος της μονάδας, που υποδηλώνει ότι η αξία μιας τιμής μειώνεται με κάθε βήμα που περνάει και έτσι αυτή πρέπει να επιτευχθεί το συντομότερο δυνατό, δηλαδή με τιε λιγότερες δυνατές κινήσεις. Τέλος, το r είναι η τιμή με την οποία ανταμείβουμε ή τιμωρούμε τον πράκτορά μας σε κάθε βήμα. Η τιμή του r είναι μηδέν για όλες τις καταστάσεις, εκτός από τις τερματικές στις οποίες λαμβάνει τιμή ανάλογα με το αποτέλεσμα του παιχνιδιού. Στις τερματικές καταστάσεις εφόσον δεν υπάρχει επόμενη κατάσταση ο όρος $\gamma Eval(s')$ παραλείπεται.

2.6.2 Μάθηση χρονικών διαφορών με χρήση ελαχίστων τετραγώνων (LSTD)

Υπάρχει όμως άλλη μία προσέγγιση στη μάθησης χρονικών διαφορών που είναι γνωστή με το όνομα Least Squares Temporal Difference (LSTD). Αντί να αλλάζουμε τα βάρη σε κάθε μας βήμα, συλλέγουμε όλες τις πληροφορίες που σχετίζονται με κάθε μετάβαση σε ένα γραμμικό σύστημα, το οποίο και επιλύουμε, ανά τακτά διαστήματα, ενημερώνοντας τα βάρη [10, 11].

Για k χαρακτηριστικά, το σύστημά μας $Aw = b$, έχει πίνακα A με μέγεθος $k \times k$ και το διάνυσμα b με μέγεθος k. Αρχικά τα A και b αρχικοποιούνται στο μηδέν, και εν συνεχεία ενημερώνονται σε κάθε κίνηση ως εξής:

$$A \leftarrow A + \phi(s)(\phi(s) – \gamma\phi(s'))^T$$

$$b \leftarrow b + \phi(s)r$$

Όπου το γ είναι όπως και πριν ο παράγοντας προεξόφλησης, το r είναι η ανταμοιβή σε περίπτωση τερματικού κόμβου (για όλες τις ενδιάμεσες καταστάσεις η τιμή του παραμένει μηδέν), και το $\phi(.)$ είναι ένα διάνυσμα μήκους k με όλες τις τιμές των χαρακτηριστικών της αντίστοιχης κατάστασης. Τέλος, όπως και στην μάθηση TD, τα δείγματα $s \rightarrow s$' μπορούν να δημιουργηθούν και σε αυτήν την περίπτωση με δύο τρόπους. Στις τερματικές καταστάσεις ο όρος $\gamma\phi(s')$ παραλείπεται, εφόσον δεν υπάρχει

επόμενη κατάσταση. Ο κώδικας του LSTD δίνεται σε μορφή ψευδοκώδικα παρακάτω (Algorithm 4).

Algorithm 4 Least Squares Temporal Difference Learning

W^π:= LSTD (D, k, ϕ, γ, π)

inputs:

D: σύνολο δειγμάτων (s, r, s') που έχουν δημιουργηθεί χρησιμοποιώντας την πολιτική π

k: αριθμός συναρτήσεων βάσης

ϕ: συναρτήσεις βάσης

γ: παράγοντας προεξόφλησης

π: πολιτική της οποίας θα προσεγγιστεί η συνάρτηση αξιολόγησης

returns: W^π : οι παράμετροι της προσεγγιστικής συνάρτηση αξιολόγησης

$A \leftarrow 0$ // ($k \times k$) πίνακας

$b \leftarrow 0$ // ($k \times 1$) διάνυσμα

for all $(s, r, s') \in D$ **do**

$A \leftarrow A + \phi(s)(\phi(s) – \gamma\phi(s'))^T$

$b \leftarrow b + \phi(s)r$

end for

$W^\pi \leftarrow A^{-1}b$

return W^π

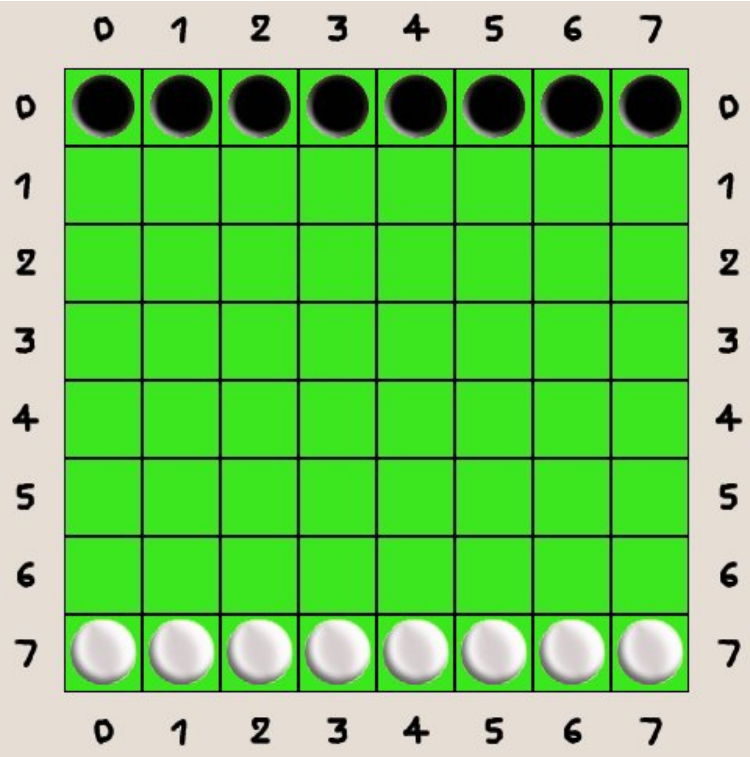
Κεφάλαιο 3

Περιγραφή του παιχνιδιού

3.1 Neighbours

Το παιχνίδι Neighbours το εμπνεύστηκε το 2003 ο Frank Riepenhausen [9]. Πρόκειται για ένα ενδιαφέρον παιχνίδι σκακιέρας με αρκετές ιδιαιτερότητες ως προς τις επιτρεπόμενες κινήσεις, που εγγυάται πολλές εναλλαγές συναισθημάτων σε παρτίδες ανθρώπου με άνθρωπο. Λόγω του υψηλού παράγοντα διακλάδωσης (μπορεί να φτάσει ακόμα και τις 64 πιθανές κινήσεις σε μια κατάσταση), αλλά και του τρόπου με τον οποίο υπολογίζονται οι επιτρεπτές κινήσεις, είναι πολύ δύσκολο για τον άνθρωπο να αντιμετωπίσει κάποιον πράκτορα επιτυχώς. Όπως θα διαπιστώσουμε όμως, η δυσκολία που αντιμετωπίζει ο άνθρωπος είναι αισθητή και στον πράκτορα, που σε ορισμένες καταστάσεις δεν μπορεί να εξερευνήσει πέρα από έξι στρώσεις σε ένα λογικό χρόνο αναμονής (λιγότερο των δύο δευτερολέπτων).

Το παιχνίδι παίζεται σε μία σκακιέρα διαστάσεων 8×8. Υπάρχουν δύο παίκτες, ο άσπρος και ο μαύρος, καθένας από τους οποίους έχει 8 πιόνια τοποθετημένα στην πρώτη και στην τελευταία γραμμή αντίστοιχα, με την πρώτη κίνηση να ανήκει στον άσπρο (Σχήμα 3.1). Οι επιτρεπτές κινήσεις είναι που αυξάνουν την δυσκολία, αλλά και το ενδιαφέρον σε αυτό το παιχνίδι. Κάθε παίκτης, όταν είναι σειρά του να παίξει, πρέπει να κάνει ακριβώς μία κίνηση. Ως κίνηση ορίζεται η ορθογώνια (κάθετη ή οριζόντια) ή διαγώνια κίνηση ενός πιονιού του παίκτη που παίζει. Όμως, η κίνηση αυτή πρέπει να είναι προς μία κατεύθυνση και να είναι ακριβώς τόσα τετράγωνα, όσο είναι το πλήθος των πιονιών (ακόμα και αντίπαλων) που βρίσκονται σε «γειτονικές» θέσεις από το πιόνι που πρόκειται να κινηθεί. Ως «γειτονικές» θέσεις ορίζουμε τα 8 (το πολύ) τετράγωνα που εντοπίζονται γύρω από το τετράγωνο που περιέχει το πιόνι που μας

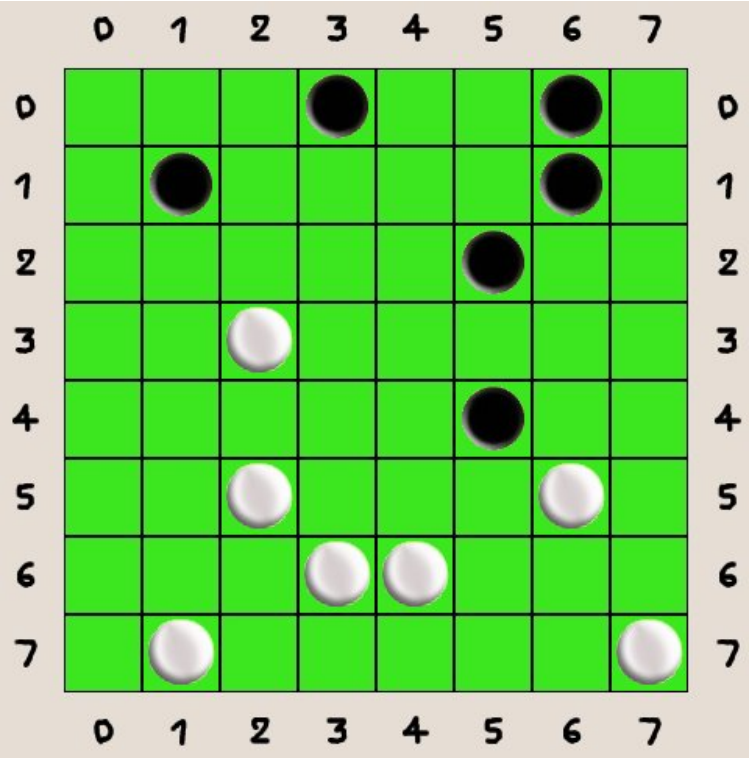


Σχήμα 3.1: Αρχική κατάσταση

απασχολεί. Όλα τα τετράγωνα που παρεμβάλλονται μεταξύ του πιονιού και του τετραγώνου προορισμού επιβάλλεται να είναι κενά, αλλιώς η κίνηση δεν είναι νόμιμη. Αν το τετράγωνο προορισμού περιέχει δικό μας πιόνι, τότε η κίνηση και πάλι δεν είναι νόμιμη, αν όμως στο τετράγωνο προορισμού βρίσκεται πιόνι του αντιπάλου, τότε το πιόνι του αντιπάλου «αιχμαλωτίζεται». Με το όρο «αιχμαλωτίζεται», εννοούμε την απομάκρυνση του αντίπαλου πιονιού από την σκακιέρα του παιχνιδιού και την κατάληψη του τετραγώνου προορισμού από το πιόνι του επιτιθέμενου, όπως θα συνέβαινε δηλαδή και σε περίπτωση που το πιόνι του αντιπάλου δεν βρισκόταν ποτέ σε εκείνο το σημείο.

Στόχος του παιχνιδιού είναι να αφήσεις τον αντίπαλο χωρίς νόμιμη κίνηση. Στο παιχνίδι όμως υπάρχουν και δύο περιπτώσεις ισοπαλίας. Ένα παιχνίδι λήγει ισόπαλο όταν εμφανιστεί, σε οποιεσδήποτε στιγμές του παιχνιδιού, τρεις φορές η ίδια κατάσταση ή περάσουν 100 συνεχόμενες στρώσεις (50 από κάθε παίκτη) χωρίς αιχμαλώτιση. Δύο καταστάσεις θεωρούνται ίδιες, όταν έχουν ακριβώς την ίδια διάταξη στη σκακιέρα και σε αυτήν την σκακιέρα καλείται να παίξει ο ίδιος παίκτης.

Στο Σχήμα 3.1 βλέπουμε την αρχική κατάσταση του παιχνιδιού. Τα πιόνια του μαύ-

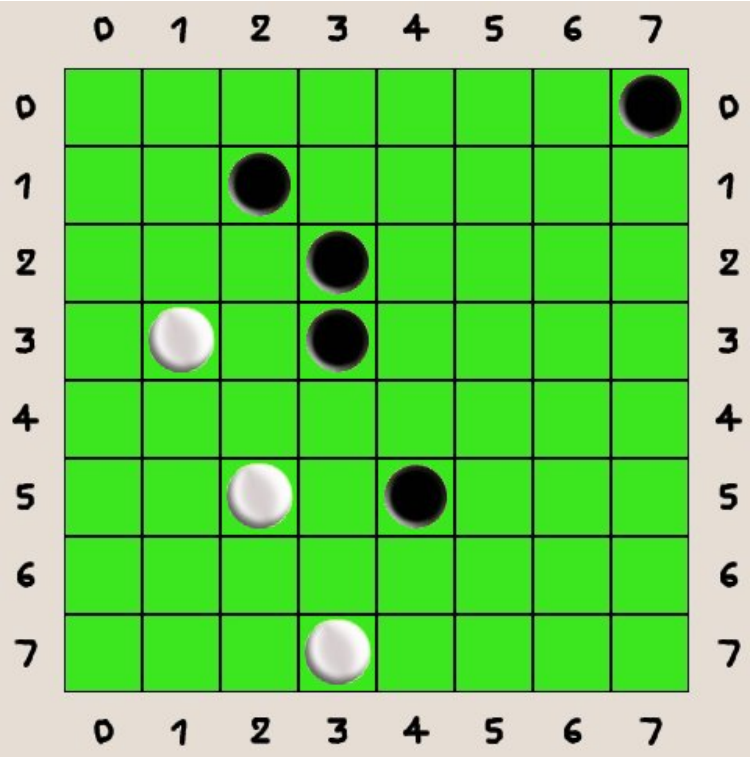


Σχήμα 3.2: Ενδιάμεση κατάσταση

ρου καταλαμβάνουν την πρώτη σειρά, ενώ εκείνα του άσπρου την τελευταία. Ο παίκτης που ξεκινάει πρώτος είναι ο άσπρος.

Στο Σχήμα 3.2 μπορούμε να παρατηρήσουμε μια ενδιάμεση κατάσταση μιας παρτίδας. Σε αυτήν την κατάσταση, ο μαύρος παίκτης έχει 21 κινήσεις διαθέσιμες, ενώ ο άσπρος 25. Τις περισσότερες κινήσεις τις έχουν τα πιονάκια στα τετράγωνα $(4, 5)$ και $(5, 6)$ καθώς μπορούν να κινηθούν μία θέση προς όλες τις κατευθύνσεις, δίνοντας έτσι το καθένα 8 επιπλέον επιλογές στους κατόχους τους. Μία λιγότερη επιλογή έχουν τα πιονάκια στις θέσεις $(5, 2)$, $(6, 4)$ και $(2, 5)$. Επίσης υπάρχουν και δύο πιονάκια, στις θέσεις $(6, 3)$ και $(1, 6)$, που γειτονεύουν με δύο πιονάκια έκαστο και έτσι είναι υποχρεωμένα, αν κινηθούν, να προχωρήσουν ακριβώς δύο τετράγωνα.

Στο Σχήμα 3.3 βλέπουμε μια τερματική κατάσταση. Είναι σειρά του άσπρου να επιλέξει την κίνησή του, αλλά δεν έχει καμία διαθέσιμη.



Σχήμα 3.3: Τερματική κατάσταση

3.2 Ο στόχος μας

Στόχος μας είναι η δημιουργία ενός πράκτορα που θα παίζει το παιχνίδι «Neighbours», χρησιμοποιώντας σύγχρονες μεθόδους αναζήτησης και εκμεταλλευόμενος την χρήση μάθησης χρονικών διαφορών. Ειδικότερα για την επιλογή της κίνησής του, ο πράκτορας μας θα εκμεταλλεύεται την ύπαρξη hash table και τις δυνατότητες που του προσφέρει ο PVS (Principal Variation Search), ενώ για την μάθηση θα χρησιμοποιηθεί η μέθοδος χρονικών διαφορών, τόσο στην απλή της μορφή (Temporal Difference), όσο και με την χρήση ελαχίστων τετραγώνων (Least Squares Temporal Difference). Για να φτάσουμε όμως στον βασικό μας στόχο πρέπει πρώτα να υλοποιηθεί το παιχνίδι καθεαυτό, καθώς δεν υπάρχει κανένα προγενέστερο περιβάλλον προσομοίωσης του παιχνιδιού και των κανόνων του. Θα προχωρήσουμε λοιπόν, στην δημιουργία ενός server-client συστήματος, με στόχο την υλοποίηση και μάθηση του παίκτη μας. Μετά την ολοκλήρωση αυτού, θα προβούμε στην δημιουργία ενός server με γραφικό περιβάλλον που θα επιτρέπει στον χρήστη να παίζει ενάντια σε άλλους χρήστες ή ενάντια σε πράκτορες που θα συνδέονται στον server μέσω του δικτύου ή ακόμα και να παρακολουθήσει ένα παιχνίδι μεταξύ δύο πρακτόρων. Τόσο οι servers όσο και ο πράκτοράς

μας πρέπει να τηρούν όλους τους κανόνες του παιχνιδιού και να παρακολουθούν με ακρίβεια την εξέλιξη του παιχνιδιού και ιδιαίτερα τις περιπτώσεις ισοπαλίας.

3.3 Σχετικές εργασίες

Όπως προαναφέραμε, δεν υπάρχει κάποια σχετική έρευνα ή εφαρμογή μεθόδων αναζήτησης και μάθησης στο παιχνίδι Neighbours. Άλλωστε αυτός ήταν και ένας από τους λόγους της επιλογής του.

Υπάρχουν όμως αρκετές επιτυχημένες εφαρμογές της ενισχυτικής μάθησης σε διάφορα παιχνίδια. Μια από τις πρώτες και σπουδαιότερες είναι η αυτή του Arthur Samuel στο παιχνίδι της ντάμας την δεκαετία του 50. Ο Samuel κατάφερε να δημιουργήσει έναν (ανώτερο του μέτριου) παίκτη για το παιχνίδι της ντάμας, που μάθαινε παίζοντας παιχνίδια κυρίως με τον εαυτό του [12]. Την κεντρική ιδέα του Samuel υιοθέτησαν αρκετοί μεταγενέστεροι πράκτορες με χαρακτηριστικότερο παράδειγμα αυτό του πράκτορα TD-Gammon [7]. Τον πράκτορα αυτόν δημιούργησε ο Gerry Tesauro το 1992 και χρησιμοποιώντας την μάθηση χρονικών διαφορών κατόρθωσε, έπειτα από αρκετές εκατοντάδες χιλιάδες παιχνίδια με τον εαυτό του, να θεωρείται ένας από τους καλύτερους παίκτες στο τάβλι. Η επιτυχία του ήταν τόσο μεγάλη που οι επιλογές του πράκτορα άλλαξαν καθιερωμένες απόψεις για το παιχνίδι και οι κινήσεις του αναγνωρίστηκαν από κορυφαίους παίκτες ως βέλτιστες.

Κεφάλαιο 4

Η δική μας προσέγγιση

4.1 Αναζήτηση

Η αναζήτησή μας βασίζεται στη λογική του αλγορίθμου MiniMax. Κατά μήκος δηλαδή του δέντρου αναζήτησης εναλλάσσονται κινήσεις μεταξύ του Min και του Max. Ακολουθώντας τη λογική της αναζήτησης με υπαναχώρηση, δημιουργούμε μόνο μία δομή κατάστασης, την οποία και μεταβάλουμε καθώς διατρέχουμε το δέντρο, προσαρμόζοντάς την πάντα στην εκάστοτε κατάσταση που ερευνούμε. Αυτό επιτυγχάνεται με την δυνατότητα μας να μπορούμε, όχι μόνο να οδηγούμαστε σε νέα σκακιέρα παίζοντας μία κίνηση, αλλά και να γυρίζουμε σε μία προηγούμενη, αναιρώντας την τελευταία κίνηση. Με αυτόν τον τρόπο δεν σώζουμε το δέντρο του παιχνιδιού στη μνήμη, παρά μόνο τον τρέχοντα κόμβο καθώς το διατρέχουμε και γλιτώνουμε από τις όποιες πράξεις αντιγραφής που θα είχαμε σε αντίθετη περίπτωση, κερδίζοντας έτσι και σε χρόνο.

Με τη χρήση Hash table σώζουμε τις ακριβείς τιμές ή τα cut-off όρια των κόμβων που εξετάζουμε, ώστε να χρησιμοποιηθούν για να αλλάξουν τα όρια σε κάποια νέα αναζήτηση, για να ταξινομήσουν τους κόμβους-παιδιά και γενικά για να μας γλιτώσουν από την αξιολόγηση καταστάσεων που έχουν ήδη αξιολογηθεί και αποθηκευτεί. Χρησιμοποιούμε για την υλοποίηση του, δύο $32 - bit$ Zobrist αριθμούς (πράγμα που αποτελεί συνήθη τακτική), λόγω κυρίως του πλεονεκτήματός τους να τροποποιούνται εύκολα με κάθε κίνηση. Έτσι η επιλογή μας να διατρέχουμε το δέντρο αναζήτησης με αυτόν τον τρόπο (εκτελώντας κινήσεις και μετά ανακαλώντας τις), βρίσκει σύμμαχο το Zobrist hashing, που με απλές πράξεις XOR, δίνει εύκολα τις αντίστοιχες διευθύνσεις του hash table για εισαγωγή αλλά και εξαγωγή δεδομένων.

Το μεγαλύτερο πλεονέκτημα που αποκομίζουμε από την χρήση του Hash table

είναι η ύπαρξη πληροφορίας που μπορεί να χρησιμοποιηθεί για να διατάξουμε τους κόμβους-παιδιά. Πριν την επέκταση των παιδιών ενός κόμβου, συμβουλευόμαστε τις όποιες τιμές του Hash table και αλλάζουμε τη σειρά των παιδιών διατάσσοντας τα σε αύξουσα ή φθίνουσα διάταξη, ανάλογα με το αν βρισκόμαστε σε κίνηση του Min ή του Max, καταφέρνοντας έτσι οι πιθανότατα καλύτεροι κόμβοι να εξετάζονται πρώτοι. Σε περίπτωση μη ύπαρξης αποθηκευμένης τιμής κάποιου κόμβου στο Hash table, πράγμα το οποίο ισχύει σε όλους τους νέους κόμβους που δημιουργούνται σε κάθε αναζήτηση (οι τελευταίες δύο στρώσεις δηλαδή), τότε χρησιμοποιούνται τυχαίοι αριθμοί για τη διάταξή τους.

Με το να χρησιμοποιούμε επαναληπτική εκβάθυνση, γεμίζουμε γρήγορα το Hash table με χρήσιμες τιμές, ακόμα και αν ο αντίπαλός μας αποφάσισε να ακολουθήσει μονοπάτι που είχαμε κλαδέψει, και κατά συνέπεια δεν είχαμε καταστάσεις του συγκεκριμένου μονοπατιού αποθηκευμένες.

Χρησιμοποιούμε άλφα-βήτα κλάδεμα και PVS ώστε να εκμεταλλευτούμε στο έπακρο τη διάταξη των κόμβων-παιδιών, μειώνοντας έτσι το συνολικό πλήθος των κόμβων.

4.2 Συνάρτηση Αξιολόγησης

Για συνάρτηση αξιολόγησης, χρησιμοποιούμε μια σταθμισμένη γραμμική συνάρτηση. Έχουμε ένα βασικό σετ που αποτελείται από 15 χαρακτηριστικά (features) που οεριγράφονται στον πίνακα 4.1. Και τα 15 χαρακτηριστικά κανονικοποιούνται κατάλληλα ώστε η τιμή τους να κυμαίνεται στο διάστημα [0, 1].

Με τα βασικά χαρακτηριστικά δεν μπορούμε να εντοπίσουμε πιθανές εξαρτήσεις μεταξύ τους. Σε αυτό θα μας βοηθήσουν τα cross terms. Πρόκειται στην ουσία για σύνθετα χαρακτηριστικά που συνδυάζουν δύο από τα βασικά, και εντοπίζουν περιπτώσεις στις οποίες έχει σημασία η στάθμιση του λόγου δύο χαρακτηριστικών.

Η χρήση των cross terms γίνεται πολλαπλασιάζοντας την τιμή του πρώτου χαρακτηριστικού με το ένα μείον την τιμή του δεύτερου. Δηλαδή:

$$crosstermvalue = featureValue1 \times (1 – featureValue2)$$

Συνολικά δημιουργήσαμε τρία σετ από χαρακτηριστικά. Αυτά τα σετ προέκυψαν από το συνδυασμό των βασικών χαρακτηριστικών με διαφορετικά cross terms. Τα αποτελέσματα τους θα συγκριθούν με στόχο να προκύψει το καλύτερο σύνολο χαρακτηριστικών. Τα χαρακτηριστικά threatsForMe και threatsForOther είναι χαρακτηριστικά που

Πίνακας 4.1: Χαρακτηριστικά κατάστασης για το παιχνίδι Neighbours.

Όνομα	Επεξήγηση
piecesForMe	Ο αριθμός των πιονιών του παίκτη μας
piecesForOther	Ο αριθμός των πιονιών του αντιπάλου
movesLeftForMe	Ο αριθμός των εναπομεινουσών κινήσεων για τον παίκτη μας στην τρέχουσα κατάσταση
movesLeftForOther	Ο αριθμός των εναπομεινουσών κινήσεων για τον αντίπαλό μας στην τρέχουσα κατάσταση
alonesForMe	Ο αριθμός των πιονιών μας που δεν έχουν πιόνι στα 8 γειτονικά τους τετράγωνα και κατά συνέπεια δεν έχουν καμία νόμιμη κίνηση
alonesForOther	Ο αριθμός των πιονιών του αντιπάλου μας που δεν έχουν πιόνι στα 8 γειτονικά τους τετράγωνα και κατά συνέπεια δεν έχουν καμία νόμιμη κίνηση
dualForMe	Ο αριθμός των δυάδων μας (πιόνια μέ έναν ακριβώς φιλικό γείτονα)
dualForOther	Ο αριθμός των δυάδων που έχει ο αντίπαλος
withOneNextForMe-2*dualForMe	Πρόκειται για τον αριθμό των πιονιών μας που γειτονεύουν με ένα πιόνι, το οποίο δεν είναι μέλος δυάδας
withOneNextForOther-2*dualForOther	Πρόκειται για τον αριθμό των ακριανών πιονιών του αντιπάλου που γειτονεύουν με ένα πιόνι, το οποίο δεν είναι μέλος δυάδας
medianDistanceForMe	Η μέση απόσταση των πιονιών μας από το κέντρο της σκακιέρας
medianDistanceForOther	Η μέση απόσταση των πιονιών του αντιπάλου από το κέντρο της σκακιέρας
onlyEnemiesForMe	Ο αριθμός των πιονιών μας που γειτονεύουν μόνο με εχθρικά
onlyEnemiesForOther	Ο αριθμός των πιονιών του αντιπάλου που γειτονεύουν μόνο με δικά μας
distanceFromLastEat	Ο κβαντισμένος σε δεκάδες αριθμός των κινήσεων που έχουν παρέλθει από την τελευταία αιχμαλώτιση

Πίνακας 4.2: Cross Terms για το 1ο σετ Χαρακτηριστικών.

Cross-terms πρώτου σετ	Χαρακτηριστικό 1	Χαρακτηριστικό 2
Cross-term 1	threatsForMe	piecesForOther
Cross-term 2	threatsForOther	piecesForMe

Πίνακας 4.3: Cross Terms για το 2ο σετ Χαρακτηριστικών.

Cross-terms δεύτερου σετ	Χαρακτηριστικό 1	Χαρακτηριστικό 2
Cross-term 1	threatsForMe	piecesForOther
Cross-term 2	threatsForOther	piecesForMe
Cross-term 3	alonesForMe	movesLeftForMe
Cross-term 4	alonesForOther	movesLeftForOther
Cross-term 5	onlyEnemiesForMe	piecesForOther
Cross-term 6	onlyEnemiesForOther	piecesForMe

χρησιμοποιήθηκαν μόνο στην δημιουργία cross terms και είναι ίσα με τον αριθμό των πιονιών μας που απειλεί ο αντίπαλος, και τον αριθμό των πιονιών του αντιπάλου που απειλούμε εμείς, αντίστοιχα.

Στους πίνακες 4.2, 4.3 και 4.4 φαίνονται τα cross terms που προστέθηκαν στα βασικά χαρακτηριστικά για να δημιουργήσουμε τα τρία σετ.

4.3 Μοντέλο εκμάθησης

Η μάθηση γίνεται με τον πράκτορά μας να παίζει παρτίδες απέναντι σε ένα του στιγμιότυπο με διαφορετικές τιμές για τα βάρη και με εναλλαγή των πλευρών μετά το πέρας κάθε παιχνιδιού. Πώς όμως επιλέγει κανείς αυτόν τον αντίπαλο; Δοκιμάστηκαν αρκετά μοντέλα κατά την διάρκεια αυτής της εργασίας, τα οποία όμως για διάφορους λόγους εγκαταλείφθηκαν. Αναφέρονται ενδεικτικά και εν συντομία κάποια από αυτά.

Μία προσέγγιση που επιχειρήθηκε ήταν η χρήση ενισχυτικής μάθησης ενάντια σε

Πίνακας 4.4: Cross Terms για το 3ο σετ Χαρακτηριστικών.

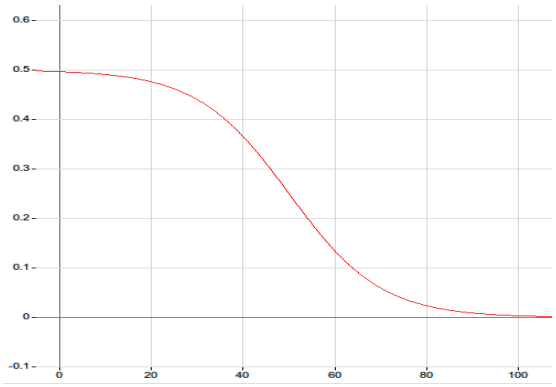
Cross-terms τρίτου σετ	Χαρακτηριστικό 1	Χαρακτηριστικό 2
Cross-term 1	threatsForMe	piecesForOther
Cross-term 2	threatsForOther	piecesForMe
Cross-term 3	dualForOther	piecesForOther
Cross-term 4	dualForMe	piecesForMe
Cross-term 5	withOneNextForOther-2*dualForOther	piecesForOther
Cross-term 6	withOneNextForMe-2*dualForMe	piecesForMe

έναν σταθερό αντίπαλο. Αυτό που παρατηρήθηκε ήταν ότι, όταν ο αντίπαλος ήταν σχετικά αδύναμος και ο παίκτης μπορούσε να πάρει κάποιες νίκες, δεν αργούσε η στιγμή που η μάθηση βελτίωνε τον πράκτορά μας, με αποτέλεσμα από ένα σημείο και έπειτα να έχουμε μόνο νίκες. Το πρόβλημα με αυτή την προσέγγιση είναι ότι ο παίκτης μας μαθαίνει μέχρι ένα σημείο, αλλά μετά δεν αποκτά νέα γνώση, λόγω της στασιμότητας του αντιπάλου. Στην ουσία, εκμεταλλευόταν ατέλειες στην στρατηγική του αντιπάλου και τίποτα παραπάνω. Επίσης, όταν ο σταθερός αντίπαλος, ήταν πολύ δυνατός, ο πράκτοράς μας δεν κέρδιζε ουσιαστικά ποτέ με αποτέλεσμα να μην μπορεί να βελτιωθεί λόγω της απουσίας θετικής ενίσχυσης.

Συνεπώς μετά την παραπάνω προσέγγιση, έγινε αντιληπτό ότι για να μάθει σωστά ο πράκτορας χρησιμοποιώντας ενισχυτική μάθηση, πρέπει να λαμβάνει όσο το δυνατό περισσότερες φορές «ενίσχυση». Πρέπει δηλαδή να βρεθεί ένας μηχανισμός, ώστε ο πράκτορας να αντιμετωπίζει πάντα κάποιον ισοδύναμό του αντίπαλο. Έτσι θα υπάρχουν και πολλές καταστάσεις νίκης και πολλές ήττας, αλλά και ισοπαλίας. Έχοντας αυτά υπόψη ακολουθήθηκε το εξής σχήμα. Ο πράκτοράς μας έπαιζε έναν αριθμό παιχνιδιών με τον αντίπαλό του, ο οποίος αρχικά είχε τα ίδια βάρη με αυτόν. Στην συνέχεια γινόταν ένα «τουρνουά» δύο παιχνιδιών, στο οποίο, αν ο πράκτορας της μάθησης κέρδιζε και τα δύο παιχνίδια, τότε ο αντίπαλός του άλλαζε τις τιμές για τα βάρη του σε αυτές του πράκτορα μάθησης και η μάθηση συνεχιζόταν. Αν ο πράκτορας που έκανε μάθηση δεν κέρδιζε και τα δύο παιχνίδια, τότε, η μάθηση συνεχιζόταν χωρίς να αλλάξουν τα βάρη του αντιπάλου.

Αυτή η προσέγγιση απέδωσε καρπούς. Ο παίκτης διαρκώς βελτιωνόταν και οι τιμές

των βαρών συνέκλιναν. Όμως, υπήρχε προβληματισμός στο γεγονός ότι μεγάλο τμήμα του δέντρου του παιχνιδιού δεν εξερευνούνταν. Έτσι προχωρήσαμε σε μια τροποποίηση της παραπάνω μεθόδου.



Σχήμα 4.1: Σιγμοειδής συνάρτηση για τη ρύθμιση της τυχαιότητας

Η μάθηση λειτουργεί ακριβώς όπως παρουσιάστηκε προηγουμένως, με την ιδιαιτερότητα ότι ο αντίπαλός μας, ακολουθώντας μία σιγμοειδή συνάρτηση (Σχήμα 4.1), καθορίζει μία πιθανότητα να επιλέξει μία τυχαία κίνηση κάθε φορά που είναι σειρά του να παίξει. Η πιθανότητα αυτή είναι σχετικά μεγάλη στα πρώτα στάδια της μάθησης και φθίνει προς το μηδέν στα τελευταία στάδια. Προφανώς, στην έναρξη του τουρνουά των δύο παιχνιδιών, ο server μας ειδοποιεί τον client και αυτός απενεργοποιεί για την διάρκεια του τουρνουά την τυχαιότητα. Με αυτήν την παραλλαγή εξερευνούμε και άλλα τμήματα του δέντρου του παιχνιδιού, συλλέγοντας επιπλέον πληροφορίες και δείγματα από ένα ευρύτερο υποσύνολο του χώρου καταστάσεων.

4.4 Παραλλαγή LSTD

Η ανάγκη για τροποποίηση του αρχικού αλγόριθμου του LSTD προέρχεται από το γεγονός ότι η πολιτική καθορίζεται μέσα από τις τιμές που δίνονται στις καταστάσεις από την συνάρτηση αξιολόγησης και διαδίδονται στην ρίζα. Αν αυτές οι τιμές αλλάξουν, το ίδιο συμβαίνει και με την πολιτική και κατ' επέκταση είναι σημαντικό να απορριφθούν τα παλιά μας δεδομένα και να χρησιμοποιηθούν μόνο τα πιο πρόσφατα για την μάθηση.

Για αυτό το λόγο τα βάρη μας ανανεώνονται μετά το τέλος μιας περιόδου που μπορεί να διαρκέσει αρκετά βήματα αποφάσεων. Στην συνέχεια αντί να μηδενίζουμε το A και το b, μειώνουμε την βαρύτητα των περιεχομένων τους, πολλαπλασιάζοντας με μία

τιμή μ μικρότερη της μονάδας και προχωράμε στην περισυλλογή των νέων δεδομένων της καινούργιας περιόδου. Ακολουθώντας αυτό το πλάνο, καταφέρνουμε να διατηρούμε τα παλαιά δεδομένα της μάθησης, με χαμηλότερη βαρύτητα, χρησιμοποιώντας τα έτσι, μέσω της επίλυσης του γραμμικού συστήματος, για να υπολογίσουμε τα νέα μας βάρη. Η τεχνική αυτή ονομάζεται Exponential Windowing.

Ο τροποποιημένος κώδικας του LSTD ακολουθεί:

Algorithm 5 LSTD με Exponential Windowing

$w^t, A^t, b^t :=$ LSTD $(D^t, k, \phi, \gamma, w^{t-1}, A^{t-1}, b^{t-1}, \mu)$

inputs:

D^t: δείγματα (s, r, s') συλλεγμένα με χρήση της πολιτικής π^{t-1} που δημιουργήθηκε από τα w^{t-1}

k: πλήθος συναρτήσεων βάσης

ϕ: συναρτήσεις βάσης

γ: παράγοντας προεξόφλησης

w^{t-1}: παράμετροι της προσέγγισης της συνάρτησης αξιολόγησης στην περίοδο $t-1$

A^{t-1}: $(k \times k)$ πίνακας A στην περίοδο $t-1$

b^{t-1}: $(k \times 1)$ διάνυσμα b στην περίοδο $t-1$

μ : ο παράγοντας του exponential windowing

outputs:

w^t: παράμετροι της προσέγγισης της συνάρτησης αξιολόγησης στην περίοδο t

A^t: πίνακας A στην περίοδο t

b^t: πίνακας b στην περίοδο t

if $t == 0$ **then**
  $A^t \leftarrow 0$
  $b^t \leftarrow 0$
else
  $A^t \leftarrow \mu A^{t-1}$
  $b^t \leftarrow \mu b^{t-1}$
end if
for all samples $(s, r, s') \in D$ **do**
  $A^t \leftarrow A^t + \phi(s)(\phi(s) - \gamma\phi(s'))^\top$
  $b^t \leftarrow b^t + \phi(s)r$
end for
$w^t \leftarrow A^{t^{-1}} b^t$
return w^t, A^t, b^t

Κεφάλαιο 5

Θέματα υλοποίησης

5.1 Γενικές πληροφορίες

Τόσο ο πράκτορας. όσο και οι servers, δημιουργήθηκαν και εκτελούνται σε περιβάλλον Linux (Ubuntu). Ο κώδικας είναι γραμμένος σε C, λόγω των απαιτήσεων μας σε ταχύτητα, ενώ για την υλοποίηση του γραφικού περιβάλλοντος χρησιμοποιήθηκε η βιβλιοθήκη GTK+ [13]. Τέλος, για την επίλυση των γραμμικών συστημάτων που προέκυπταν λόγω του LSTD χρησιμοποιήθηκε η βιβλιοθήκη GSL [8].

5.2 Υλοποίηση πράκτορα

Για λόγους ταχύτητας, αλλά και οικονομίας μνήμης, το δέντρο αναζήτησης δεν αποθηκεύεται. Αντ' αυτού, παρέχουμε δύο συναρτήσεις doMove και undoMove και με αυτές «διατρέχουμε» το δέντρο χωρίς να χρειάζεται να κάνουμε άσκοπες αντιγραφές μνήμης και να χάνουμε πολύτιμο χρόνο. Οι συναρτήσεις αυτές είναι υπεύθυνες για τη σωστή αλλαγή της σκακιέρας μας, καθώς και για τη σωστή επεξεργασία όλων των δεδομένων που αποθηκεύονται στην δομή της σκακιέρας, έτσι ώστε να αντικατοπτρίζουν την εκάστοτε κατάσταση.

Γενικά, όπως έγινε αντιληπτό κατά την διάρκεια της υλοποίησης, μια καλή τακτική είναι να κρατάμε τιμές μέσα στη δομή της σκακιέρας, που θα μας βοηθούν σε όλες τις λειτουργίες του πράκτορά μας και είναι εύκολο να προσαρμοστούν από μία παλαιότερη τιμή σε μια νέα κατά την διάρκεια της αναζήτησης. Τέτοιες τιμές είναι: τα δύο Zobrists που χρησιμοποιούμε, η απόσταση (σε κινήσεις) από την τελευταία αιχμαλώτιση και δύο πίνακες με τις θέσεις των πιονιών των δύο παικτών, καθώς είναι καλύτερο για

κάποιες λειτουργίες, όπως για παράδειγμα για την αξιολόγηση να διατρέχουμε δύο πίνακες 8 θέσεων από έναν πίνακα 64 θέσεων.

5.3 Hash Table

5.3.1 Δομή Hash Table

Το Hash table, όπως θα γίνει αντιληπτό, εξυπηρετεί πολλούς σκοπούς και η χρήση του σε όλους τους μοντέρνους πράκτορες παιχνιδιών θεωρείται δεδομένη. Οι πληροφορίες που αποθηκεύονται σε κάθε εγγραφή του hash table είναι τέσσερις. Αρχικά θα αναφερθούμε σε κάθε μία από αυτές, και στην συνέχεια θα δώσουμε μια λεπτομερή περιγραφή της λειτουργίας του hash table.

1. **Αναπαράσταση της κατάστασης**
   Πρόκειται για έναν 32-bit Zobrist αριθμό που αντιπροσωπεύει την κατάσταση.

2. **Τιμή αξιολόγησης της κατάστασης**
   Πρόκειται για έναν float με την ακριβή τιμή αξιολόγησης της κατάστασης ή την τιμή του cutoff με το οποίο σταμάτησε η αναζήτηση σε εκείνο τον κόμβο.

3. **Βάθος**
   Πρόκειται για το βάθος από το οποίο προέκυψε η τιμή. Όσο μεγαλύτερο είναι το βάθος, προφανώς τόσο πιο πολύτιμη είναι αυτή η εγγραφή στο hash. Όπως θα δούμε στην λειτουργία του hash, χρησιμοποιείται για να αποφασίσουμε αν η τιμή που έχει αποθηκευτεί μας ενδιαφέρει.

4. **Τύπος**
   Πρόκειται για τον τύπο της τιμής αξιολόγησης. Συγκεκριμένα, ο τύπος έχει την τιμή 1, αν πρόκειται για ακριβή τιμή MiniMax, την τιμή 2, αν πρόκειται για lower bound και την τιμή 3, αν πρόκειται για upper bound.

5.3.2 Λειτουργία Hash Table

Το συνολικό μέγεθος μιας εγγραφής είναι 10 bytes, μέγεθος αξιοπρεπές που μας επιτρέπει να έχουμε ένα αρκετά μεγάλο hash table χωρίς ιδιαίτερες απαιτήσεις από πλευράς μνήμης. Συνολικά στην υλοποίηση του hash table μας χρησιμοποιούνται, όπως

προείπαμε, 2 Zobrist αριθμοί που σώζονται στη δομή της κατάστασης και ενημερώνονται με κάθε κίνηση. Ο ένας αριθμός ονομάζεται lock και ο άλλος key. Ο key λειτουργεί σαν index στο hash table, ενώ ο lock σώζεται κανονικά μέσα στον πίνακα. Άρα όταν θέλουμε να βρούμε αν έχουμε μια κατάσταση αποθηκευμένη στο hash αρκεί να ψάξουμε στη θέση hashTable[zobkey%HASHSIZE]. Αν το Zobrist lock είναι ίδιο στην κατάσταση που παρατηρούμε και στο hash table, τότε πιθανότατα έχουμε hash hit (υπάρχει μια πραγματικά πολύ μικρή πιθανότητα να έχουμε hash error – πράγμα το οποίο δεν μπορούμε ωστόσο να επιβεβαιώσουμε), διαφορετικά έχουμε hash miss. Το βάθος παίζει σημαντικό ρόλο στην απόφασή μας να χρησιμοποιήσουμε τα δεδομένα. Η αποθηκευμένη τιμή χρησιμοποιείται μόνο όταν έχουμε hash hit και το βάθος από το οποίο υπολογίστηκε είναι μεγαλύτερο ή ίσο από το βάθος αναζήτησης που έχει ακόμα η κατάσταση αυτή στην αναζήτησή μας. Ανάλογα με τον τύπο της τιμής, η τιμή επιστρέφεται ως τιμή του κόμβου (αν είναι ακριβής τιμή) ή απλά ενημερώνει το α ή το β της αναζήτησης με κλάδεμα άλφα-βήτα.

5.3.3 Hash Table και κλάδεμα άλφα-βήτα

Με το να αποθηκεύουμε τα cutoff όταν αυτά προκύπτουν, παρέχουμε στον αλγόριθμο αναζήτησης μια επιπλέον πληροφορία προς εκμετάλλευση. Αφού βεβαιωθούμε ότι έχουμε hash hit και βάθος από το οποίο μπορούμε να δεχτούμε τα δεδομένα, μπορούμε να ενημερώσουμε τις τιμές των άλφα και βήτα με το που ξεκινάμε την εξέταση του κόμβου. Αυτή η ενημέρωση γίνεται μόνο σε περίπτωση που τα αποθηκευμένα άλφα και βήτα είναι καλύτερα από εκείνα με τα οποία έχουμε φτάσει στον επίμαχο κόμβο. Αυτό δημιουργεί μικρότερα παράθυρα άλφα-βήτα και συνεπώς περισσότερα cutoff, ώστε τελικά έχουμε λιγότερους κόμβους προς επέκταση και περισσότερο χρόνο στη διάθεση του προγράμματός μας.

5.3.4 Hash Table ως Trasposition Table

Σε πολλά παιχνίδια υπάρχει περίπτωση η ίδια κατάσταση να απαντηθεί σε διάφορα σημεία του δέντρου αναζήτησης. Αυτό συμβαίνει διότι σε μία ακολουθία κινήσεων A-B-C υπάρχει περίπτωση οι κινήσεις να μην επηρεάζονται μεταξύ τους και ως αποτέλεσμα η ακολουθία C-B-A να οδηγεί στην ίδια ακριβώς κατάσταση, δηλαδή σε μία transposed κατάσταση. Το hash table με το να σώζει τις τιμές των προηγούμενων καταστάσεων έχει την δυνατότητα να εντοπίζει αυτές τις αντιμεταθέσεις και να επιστρέφει αμέσως σε περίπτωση που έχει αποθηκεύσει κάποια ακριβή τιμή για τη συγκεκριμένη

κατάσταση.

5.3.5 Hash Table και διάταξη παιδιών

Με το να μπορούμε να ανακτήσουμε ακριβείς τιμές ή τιμές cutoff, παίρνουμε μια ιδέα από προηγούμενες αναζητήσεις για το πόσο καλή είναι μια κατάσταση, πριν ακόμα στρέψουμε την αναζήτησή μας προς τα εκεί. Μπορεί αυτή τη στιγμή να μην γίνεται αντιληπτό, αλλά αυτό είναι ίσως το σημαντικότερο πράγμα που μας προσφέρει το hash table. Με αυτήν την γνώση μπορούμε να κάνουμε μια πάρα πολύ καλή ταξινόμηση των παιδιών (move ordering) πριν ακόμα επεκταθούν, κοιτάζοντας απλά τις αποθηκευμένες τιμές μας. Η καλή ταξινόμηση των παιδιών έχει άμεση επίδραση στην αποτελεσματικότητα του PVS.

5.4 Εντοπισμός ισοπαλιών

Πάρα πολλοί πράκτορες που κυκλοφορούν για διαδεδομένα παιχνίδια, συχνά αγνοούν τις συνθήκες ισοπαλίας (εκτός και αν είναι αναπόσπαστο μέρος του παιχνιδιού) για να μην βλάψουν την ταχύτητα του παίκτη τους. Λόγω όμως των στόχων μας, όπου μεταξύ άλλων έχουμε και τη σωστή υλοποίηση του παιχνιδιού με όλες τις πτυχές του, οι ισοπαλίες δεν μπορούσαν να παραβλεφθούν.

5.4.1 Υλοποίηση του συστήματος εύρεσης ισοπαλιών

Στον κόσμο του Neighbours μπορούμε να έχουμε δύο ειδών ισοπαλίες. Η πρώτη περίπτωση είναι η εξάντληση 100 κινήσεων (50 για κάθε παίκτη) από την τελευταία κίνηση «αιχμαλώτισης», ενώ η δεύτερη είναι η γνωστή μας από το σκάκι, δηλαδή τριπλή επανάληψη της ίδιας κατάστασης σε οποιοδήποτε σημείο του παιχνιδιού.

Για τον εντοπισμό, λοιπόν, αυτών των δύο συνθηκών χρησιμοποιείται ένας πίνακας «ιστορικού». Σε αυτόν τον πίνακα σώζονται καταστάσεις του παιχνιδιού και συγκεκριμένα τρία πράγματα: ο Zobrist αριθμός key, ο Zobrist αριθμός lock και το αν είχαμε «αιχμαλώτιση» με την κίνηση με την οποία οδηγηθήκαμε σε αυτήν την κατάσταση. Έτσι, όταν δημιουργούμε το δέντρο αναζήτησης και διατρέχουμε τους κόμβους του, ταυτόχρονα ενημερώνουμε και αυτόν τον πίνακα συμπληρώνοντας και στη συνέχεια αφαιρώντας διαδοχικές καταστάσεις. Με την κάθε προσθήκη μιας καινούργιας κατάστασης γίνεται έλεγχος για το αν πληρούνται οι συνθήκες ισοπαλίας και ενημερώνεται σχετικά η δομή της κατάστασης (και πιο συγκεκριμένα η μεταβλητή drawFlag)

για το αν αυτή η κατάσταση αποτελεί τερματικό κόμβο ισοπαλίας. Ο έλεγχος γίνεται διατρέχοντας τον πίνακα των προηγούμενων καταστάσεων έως την πιο πρόσφατη αιχμαλώτιση ή μέχρι την συμπλήρωση 100 κινήσεων ή μέχρι την πρώτη επαναλαμβανόμενη κατάσταση. Όταν εισάγεται η αρχική κατάσταση θεωρείται ως κατάσταση που προήλθε από κίνηση με «αιχμαλώτιση». Αυτό βοηθάει στην σωστή μέτρηση της απόστασης από την τελευταία αιχμαλώτιση.

Όπως γίνεται εμφανές, δεν αναζητούμε για τριπλή εμφάνιση της ίδιας κατάστασης, αλλά θεωρούμε ότι έχουμε ισοπαλία με μία επανάληψη. Αυτή μας η επιλογή δεν είναι λανθασμένη, καθώς μπορούμε να θεωρήσουμε ότι το να ξαναγυρίσουμε με τις κινήσεις μας πίσω σε μία προηγούμενη κατάσταση είναι ένα βήμα πιο κοντά στην ισοπαλία και θα έπρεπε να βαθμολογείται σαν τέτοια, όπως και γίνεται. Τέλος, μέσα από την διαδικασία της επέκτασης του δέντρου αναζήτησης, ερευνάται η τιμή της μεταβλητής drawFlag της δομής κατάστασης, και παίρνονται τα απαραίτητα μέτρα, αξιολόγηση δηλαδή του κόμβου με μηδέν και επιστροφή στο γονικό κόμβο με αυτήν την τιμή.

Αξίζει σε αυτό το σημείο να γίνει μια σημαντική παρατήρηση. Τα Zobrist που σώζονται μέσα στο πίνακα ιστορικού είναι διαφορετικά από αυτά που χρησιμοποιούνται στο hash table. Διαφέρουν κατά ένα XOR του Zobrist αριθμού που συμβολίζει την απόστασή μας από την τελευταία αιχμαλώτιση. Αυτό συμβαίνει διότι, στο μεν hash table μας ενδιαφέρει να βρούμε ακριβώς την ίδια σκακιέρα λαμβάνοντας υπόψη και την απόσταση από την αιχμαλώτιση, καθώς αποτελεί χαρακτηριστικό που εμπεριέχεται στην συνάρτηση αξιολόγησης. Στην περίπτωση όμως των ισοπαλιών δεν θέλουμε κάτι τέτοιο να συμβαίνει, έτσι κάνουμε μία επιπλέον XOR για να πάρουμε τον Zobrist αριθμό χωρίς την απόσταση από την αιχμαλώτιση.

5.4.2 Εντοπισμός καταστάσεων ισοπαλίας στον server

Στο παραπάνω σύστημα εντοπισμού ισοπαλιών, υπάρχει μια μικρή πιθανότητα λανθασμένης υπόδειξης. Η λανθασμένη υπόδειξη κάποιας σκακιέρας ως σκακιέρα ισοπαλίας προϋποθέτει τα ακόλουθα γεγονότα:

1. Ύπαρξη hash error (2 καταστάσεις να έχουν ακριβώς τα ίδια zobrist).
2. Στις δύο επίμαχες καταστάσεις να παίζει ο ίδιος παίκτης (παρόλο που υπάρχει zobrist αριθμός που αποτυπώνει στην κατάσταση το ποιος έχει σειρά, η σειρά επιβεβαιώνεται και από τον αλγόριθμο αναζήτησης ισοπαλιών, καθώς ελέγχει για την ύπαρξη ταυτόσημων καταστάσεων σε κάθε ζυγό βήμα).
3. Οι δύο καταστάσεις να εντοπίζονται μέσα στις τελευταίες το πολύ 50 κινήσεις.

4. Να μην υπάρχει κατάσταση με «αιχμαλώτιση» ανάμεσα στις δύο καταστάσεις.

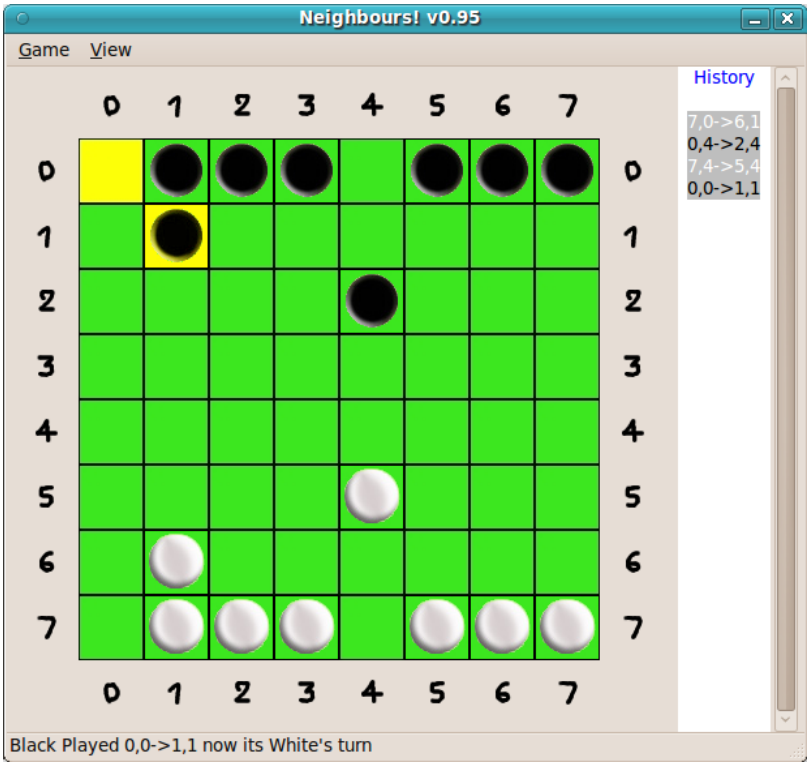
Αν και ουσιαστικά ανύπαρκτη, δεν παύει να είναι μη μηδενική η πιθανότητα να έχουμε όλα τα παραπάνω. Προφανώς, από την πλευρά του server δεν μπορούμε να δεχτούμε την παραμικρή πιθανότητα να συμβεί ένα τέτοιο λάθος και συν τοις άλλοις εδώ πρέπει να μετράμε ακριβώς για 3 ίδιες καταστάσεις για να σημειώσουμε την ισοπαλία. Είναι λοιπόν εμφανές ότι χρειαζόμαστε ένα διαφορετικό σύστημα εντοπισμού ισοπαλιών.

Το σύστημα που υιοθετήθηκε είναι παρόμοιο με αυτό του πράκτορα, έχει όμως τις ακόλουθες δύο σημαντικές διαφορές. Πρώτον, σώζουμε όλη την σκακιέρα στον πίνακα εντοπισμού (μαζί με τις αιχμαλωτίσεις) και δεύτερον, μετράμε τρεις ακριβώς επαναλήψεις για να σημειώσουμε ισοπαλία λόγω επανάληψης καταστάσεων. Με αυτό τον τρόπο, βρίσκουμε πάντα σωστά τις ισοπαλίες και εν συνεχεία ενημερώνουμε τους παίκτες ότι το παιχνίδι τελείωσε.

5.5 Γραφικό περιβάλλον

Το γραφικό περιβάλλον (Σχήμα 5.1) που υλοποιήθηκε επιτρέπει τη διεξαγωγή παρτίδων μεταξύ του πράκτορα που δημιουργήσαμε (σε διάφορα επίπεδα δυσκολίας), εξωτερικών πρακτόρων που μπορούν να συνδεθούν μέσω δικτύου, αλλά και του ανθρώπου. Ο χρήστης έχει την δυνατότητα να αλλάξει τους παίκτες κατά τη διάρκεια μιας παρτίδας, χρησιμοποιώντας οποιεσδήποτε από τις επιλογές παικτών. Η παρτίδα μπορεί να αποθηκευτεί και να επανακτηθεί, όποτε ο χρήστης το επιθυμήσει.

Οπτικά ο χρήστης υποβοηθείται με το να σημειώνονται πάνω στην σκακιέρα οι εκάστοτε επιτρεπτές κινήσεις κάθε πιονιού που επιλέγει. Οι κινήσεις της παρτίδας εμφανίζονται στην λίστα του ιστορικού και η τελευταία κίνηση επισημαίνεται με έντονο χρώμα πάνω στην σκακιέρα. Στο κάτω μέρος υπάρχει γραμμή μηνυμάτων που ενημερώνει το χρήστη για την κατάσταση του παιχνιδιού, την τελευταία κίνηση, ποιός παίζει, κ.λ.π. Όλα αυτά τα βοηθητικά χαρακτηριστικά μπορούν σε οποιαδήποτε στιγμή να απενεργοποιηθούν και να ενεργοποιηθούν ξανά από το αντίστοιχο μενού.



Σχήμα 5.1: Γραφικό περιβάλλον

Κεφάλαιο 6

Αποτελέσματα

6.1 Διαδικασία εκμάθησης

Όπως ήδη αναφέρθηκε, ως αντίπαλός μας κατά την διάρκεια της μάθησης, είναι ένα αντίγραφο του ίδιου μας του πράκτορα. Στην αρχή οι δύο παίκτες έχουν ακριβώς τα ίδια βάρη. Όμως, ενώ σε εκείνον που δεν εφαρμόζεται μάθηση, τα βάρη παραμένουν ίδια, δεν συμβαίνει το ίδιο και στον πράκτορα που μαθαίνει. Στην περίπτωση της μάθησης χρονικών διαφορών(TD), τα βάρη αλλάζουν μετά το πέρας κάθε κίνησης, ενώ με την χρήση των ελαχίστων τετραγώνων(LSTD), τα βάρη αλλάζουν σε περιόδους των 1000 κινήσεων, κάθε φορά δηλαδή που λύνεται το γραμμικό μας σύστημα για τον υπολογισμό τους.

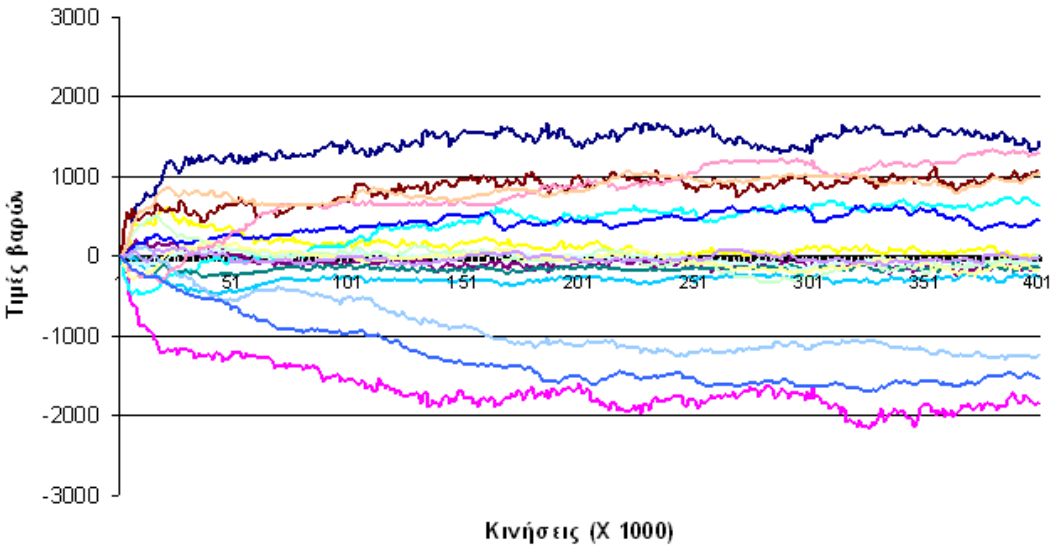
Μετά την πάροδο 50 παιχνιδιών μάθησης, αξιολογούμε την μέχρι τώρα επίδοση του πράκτορά μας και αποφασίζουμε αν βελτιωθήκαμε, με στόχο να αυξήσουμε την δυσκολία και να συνεχίσουμε, έχοντας πλέον αντιγράψει τα βάρη μας στον αντίπαλό μας πράκτορα (κλωνοποίηση). Αυτή η αξιολόγηση, γίνεται μέσα από δύο παιχνίδια όπου εναλλάσονται πλευρές έτσι ώστε να εξεταστεί η απόδοση του πράκτορα ανεξαρτήτως χρώματος παίκτη.

6.2 TD vs LSTD

Για να συγκρίνουμε την απόδοση της μεθόδου ενισχυτικής μάθησης με χρήση χρονικών διαφορών (TD), με την παραλλαγή της που κάνει χρήση ελαχίστων τετραγώνων (LSTD), χρησιμοποιήσαμε τις δύο μεθόδους στο ίδιο σετ χαρακτηριστικών για τον ίδιο

αριθμό παιχνιδιών. Συγκεκριμένα, χρησιμοποιήθηκε το βασικό σετ χαρακτηριστικών και η μάθηση διενεργήθηκε με 20.000 παιχνίδια.

Η διακύμανση των τιμών που είχαν τα βάρη των χαρακτηριστικών καθ' όλη την διάρκεια της μάθησης και στις δύο περιπτώσεις φαίνονται στα Σχήματα 6.1 και 6.2.

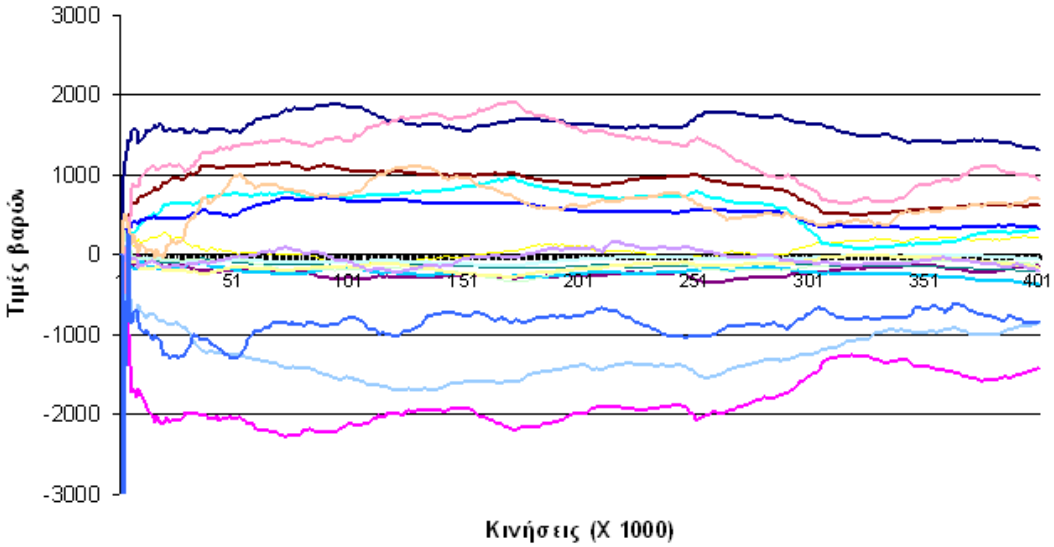


Σχήμα 6.1: TD (1ο σετ) μετά από 400.000 κινήσεις μάθησης

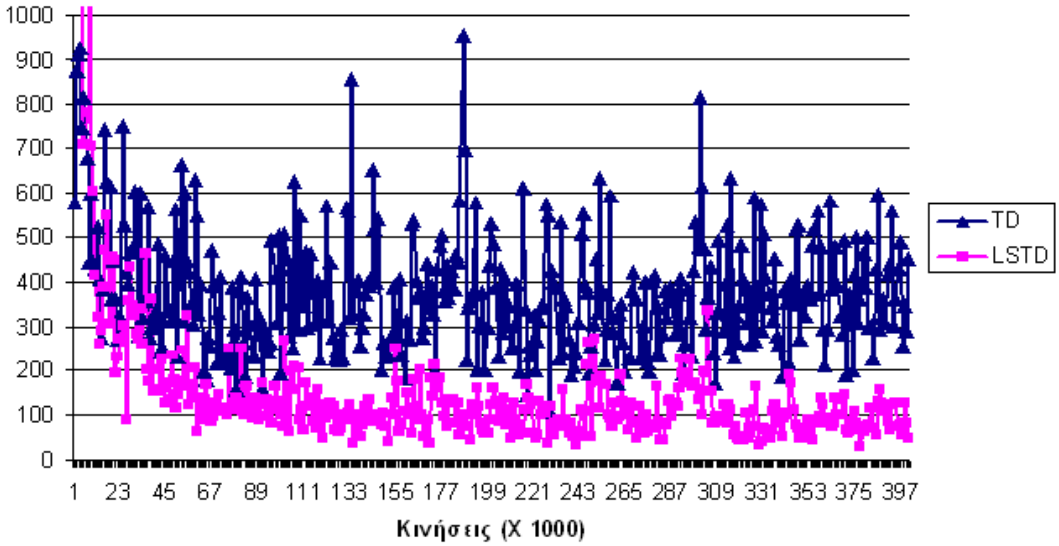
Επίσης, για να ελέγξουμε τη συγκλιση των βαρών υπολογίσαμε την L_1 norm (άθροισμα απολύτων τιμών) της διαφοράς διαδοχικών τιμών των βαρών.

$$L_t = \sum_{j=1}^{k} |w_t(j) - w_{t-1}(j)|$$

Οι καμπύλες μάθησης για τον TD και τον LSTD φαίνονται στο Σχήμα 6.3 και έτσι μπορούμε να βγάλουμε συμπεράσματα σχετικά με την σύγκλιση των δύο αλγορίθμων. Τα οφέλη του LSTD γίνονται γρήγορα αντιληπτά. Ο αλγόριθμος LSTD φαίνεται να συγκλίνει πιο γρήγορα και πιο ομαλά, χάρη στο μεγάλο του πλεονέκτημα, να μην εξαρτάται από την σειρά εισαγωγής των δεδομένων σε μια περίοδο, όπως ο TD. Τα βάρη τα οποία προέκυψαν έπειτα από την εκτέλεση του LSTD οδήγησαν σε έναν δυνατότερο πράκτορα σε σύγκριση με εκείνον που ήταν αποτέλεσμα της μάθησης με TD, καθώς σε σύνολο 100 παιχνιδιών ο πρώτος κέρδισε 60 αγώνες, ενώ ο δεύτερος 30, και 10 αγώνες έληξαν ισόπαλοι.



Σχήμα 6.2: LSTD (1ο σετ) μετά από 400.000 κινήσεις μάθησης



Σχήμα 6.3: Καμπύλες μάθησης για TD και LSTD

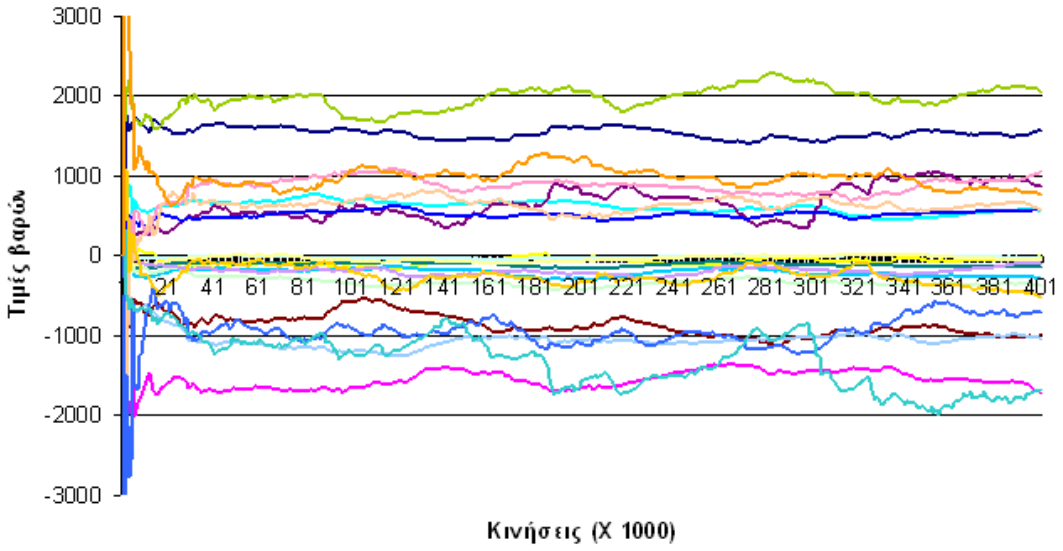
6.3 Τα τρία σετ χαρακτηριστικών

Παραπάνω παρατηρήσαμε την συμπεριφορά του πρώτου σετ χαρακτηριστικών, και επισημάναμε την ανωτερότητα του LSTD. Λόγω αυτής της ανωτερότητας έναντι του

Πίνακας 6.1: Αποτελέσματα τουρνουά 100 αγώνων μεταξύ των τριών σετ.

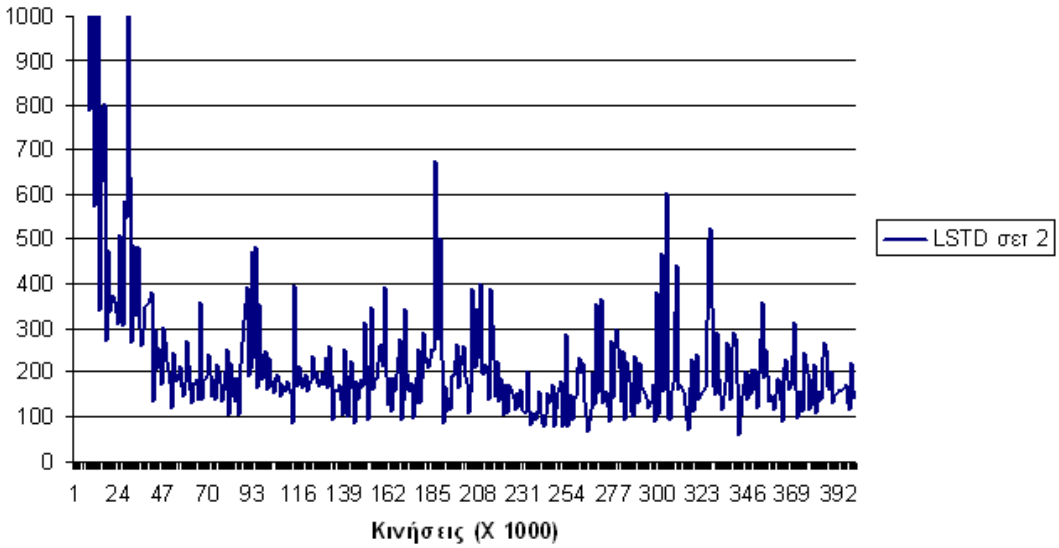
	Πρώτο σετ	Δεύτερο σετ
Νίκες	44	49
	Πρώτο σετ	Τρίτο σετ
Νίκες	48	41
	Δεύτερο σετ	Τρίτο σετ
Νίκες	46	43

TD, στα άλλα δύο σετ χαρακτηριστικών παραθέτουμε τα αποτελέσματα μόνο της μεθόδου χρονικών διαφορών με χρήση ελαχίστων τετραγώνων.

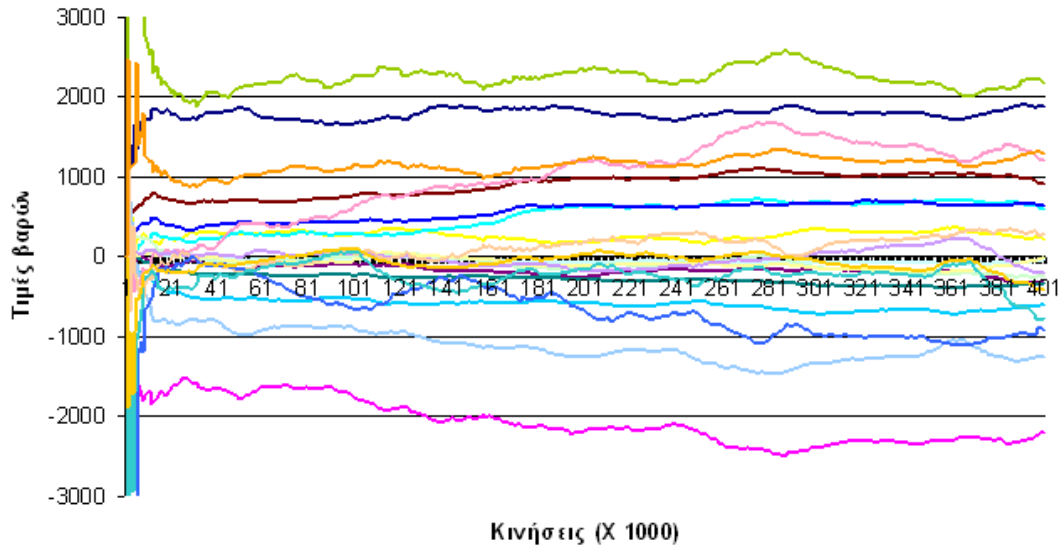


Σχήμα 6.4: LSTD (2ο σετ) μετά από 400.000 κινήσεις μάθησης

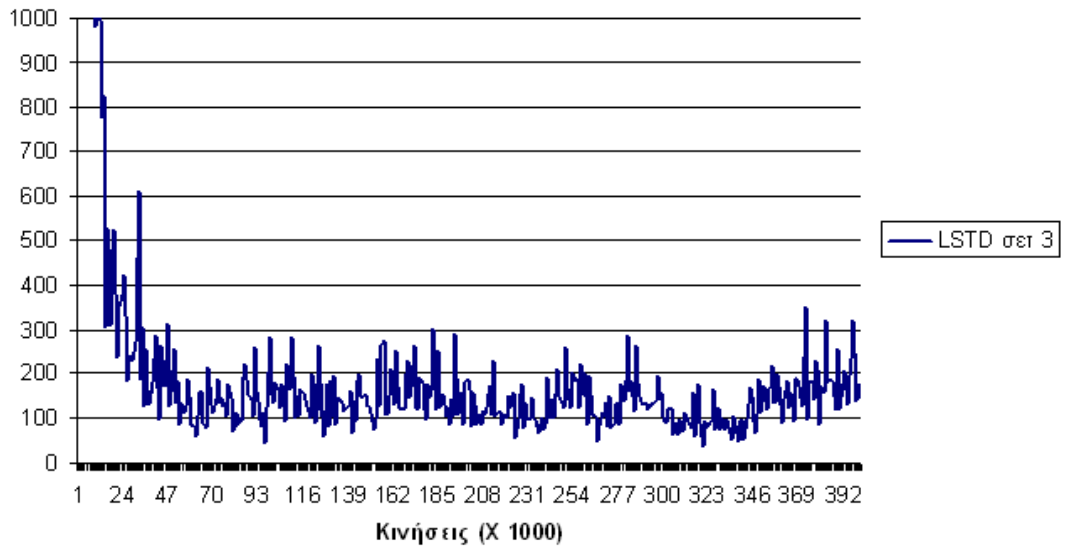
Έπειτα από ένα μικρό τουρνουά των 100 παιχνιδιών ανάμεσα στους πράκτορες που προέκυψαν από τα 3 σετ χαρακτηριστικών, έγινε εμφανές ότι ο πιο δυνατός πράκτορας ήταν αυτός που χρησιμοποιούσε το 2ο σετ.



Σχήμα 6.5: Καμπύλη μάθησης για LSTD με το σετ 2



Σχήμα 6.6: LSTD (3ο σετ) μετά από 400.000 κινήσεις μάθησης



Σχήμα 6.7: Καμπύλη μάθησης για LSTD με το σετ 3

Κεφάλαιο 7

Συμπεράσματα

7.1 Συμπεράσματα

Ο πράκτορας που δημιουργήθηκε επιτυγχάνει αναζήτηση σε βάθος οκτώ στρώσεων, πράγμα το οποίο γίνεται εφικτό σε ένα τέτοιο παιχνίδι με τόσο μεγάλο παράγοντα διακλάδωσης (μπορεί να φτάσει και το 64), κυρίως λόγω της ύπαρξης και της πλήρους εκμετάλλευσης του hash table.

Το παιχνίδι είναι πολύ απρόβλεπτο, και ο άνθρωπος δύσκολα μπορεί να αντιμετωπίσει με επιτυχία τον υπολογιστή. Λόγω αυτών των απότομων αλλαγών που παρατηρούνται, έγινε εμφανής η σημασία που θα είχε ένα όσο το δυνατό μεγαλύτερο βάθος αναζήτησης. Αυτός ήταν και ο λόγος που ο πράκτορας δημιουργήθηκε με στόχο την ταχύτητα. Η χρήση ελαχίστων τετραγώνων με την μέθοδο χρονικών διαφορών, οδήγησε σταδιακά σε αρκετά καλούς παίκτες, ακολουθώντας το επιτυχημένο μοντέλο μάθησης που ακολουθήσαμε.

Ένας επίσης από τους δευτερεύοντες στόχους ήταν και η όσο το δυνατό μικρότερη κατανάλωση πόρων του συστήματος. Στόχος ο οποίος και επετεύχθει με την χρήση αναζήτησης με υπαναχώρηση, με το πρόγραμμα να καταλαμβάνει συνολικά μόνο 11MB μνήμη όταν εκτελείται, το σύνολο της οποίας χρησιμοποιούν κατά κανόνα οι 1.000.000 εγγραφές του hash table.

7.2 Μελλοντικές βελτιώσεις

Μεταξύ των βελτιώσεων που θα μπορούσε να δεχτεί αυτή η εργασία είναι και η χρήση κάποιου αλγόριθμου παραλληλισμού για την πλήρη εκμετάλλευση της υπολογιστικής

ισχύος των σύγχρονων επεξεργαστών. Η επιπλέον αυτή ισχύς θα μπορούσε να δυναμώσει τον πράκτορα και να επιτρέψει την χρήση του κερδισμένου χρόνου για τη βελτίωση των επιλογών του.

Επίσης, θα μπορούσε το σύνολο των χαρακτηριστικών να εμπλουτιστεί, οδηγώντας έτσι στην βελτίωση του πράκτορα, έπειτα από νέα εφαρμογή μάθησης. Όπως τέλος, θα μπορούσε να εξεταστεί και η απόδοση διαφορετικών αλγορίθμων αναζήτησης, όπως για παράδειγμα ο MTD(f).

Βιβλιογραφία

[1] Claude E. Shannon, Programming a Computer for Playing Chess. Philosophical Magazine, Ser.7, Vol. 41, No. 314 – March 1950

[2] Stuart Russel and Peter Norvig, Artificial Intelligence: A Modern Approach. Prentice Hall, 2 ed., 2003.

[3] T. Anthony Marsland, Murray Campbell (1982). Parallel Search of Strongly Ordered Game Trees. ACM Computing Survey 14(4): 533-551

[4] Richard S. Sutton and Andrew G. Barto, Reinforcement Learning:An Introduction. The MIT Press, 1 ed., 1998.

[5] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," Journal of Artificial Intelligence Research vol. 4, pp. 237–285, 1996.

[6] A.L. Zobrist. A New Hashing Method with Applications for Game Playing. ICCA Journal, 13(2):69–73, 1990.

[7] G. Tesauro. Temporal Difference Learning and TD-Gammon. Communications of the ACM, 38(3), March 1995.

[8] GNU Scientific Library (GSL) http://www.gnu.org/software/gsl/

[9] 2003 Frank Riepenhausen superdupergames.org/rules/neighbours.pdf

[10] S. J. Bradtke, A. G. Barto, and P. Kaelbling, "Linear least-squares algo- rithms for temporal difference learning," in Machine Learning, pp. 22–33, 1996.

[11] J. A. Boyan, "Least-squares temporal difference learning," in In Proceedings of the Sixteenth International Conference on Machine Learning, pp. 49–56, 1999.

[12] Samuel, A. L. (1959). Some studies in machine learning using the game checkers. IBM Journal of Research and Development, 3(3), 210-229

[13] The GTK+ (GIMP Toolkit) project http://www.gtk.org/