Technical
University
of Crete

Τεχνητή Νοημοσύνη

Αναφορά 2ης Εργαστηριακής άσκησης

Φοιτητές

Στέφανος Καλογεράκης | 2015030064

Διδάσκων

Χαλκιαδάκης Γ.

Εισαγωγή

Στην δεύτερη προγραμματιστική άσκηση του μαθήματος κληθήκαμε να σχεδιασουμε και να υλοποιήσουμε ένα πρόγραμμα που παίζει το παιχνίδι TUC-ANTS. Το παιχνίδι αυτό αποτελεί μια απλοποιημένη παραλλαγή της ντάμας(checkers) με τους κανόνες να περιγράφονται από την δοθείσα εκφώνηση του. Σκοπός της άσκησης είναι η εξοικείωση με τις διαφορετικές τεχνικές αναζήτησης και η εισαγωγή άλλων τεχνικών προκειμένου να παρατηρηθεί η απόδοσή τους.

Υλοποίηση

Στις παρακάτω ενότητες αναλύονται με την σειρά οι τεχνικές που δοκιμάστηκαν κατά την εκπόνηση της εργασίας καθώς και τα διαφορετικά στάδια της εκφώνησης.

Τρέχουσα δομή κώδικα

Το μεγαλύτερο μέρος των υλοποιήσεων και τεχνικών που πραγματοποιήθηκαν βρίσκονται στο αρχείο client.c σε συνέχεια του αρχικού κώδικα. Προστέθηκαν όμως κάποια νέα αρχεία προκειμένου να συμπληρώσουν ανάγκες που προέκυψαν κατά την υλοποίηση. Αρχικά δημιουργήθηκε header file **client.h** στο οποίο βρίσκεται ο ορισμός των καινούργιων συναρτήσεων που υλοποιήθηκαν. Στο αρχείο **LinkedList.c** δημιουργήθηκαν δομές τύπου LinkedList απαραίτητες για την δυναμική αποθήκευση των διαθέσιμων κινήσεων του παίκτη, ενώ σε συνέχεια δημιουργήθηκε και ένα αρχείο δοκιμών **ListTester.c** το οποίο δεν χρησιμοποιείται κάπου στην τελική υλοποίηση.

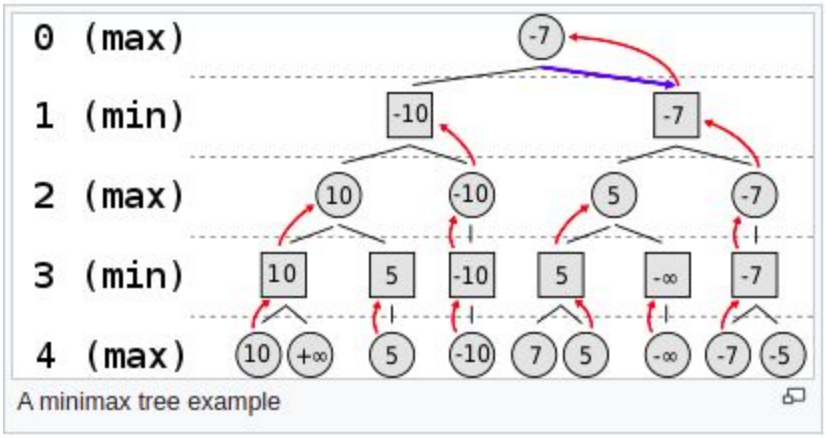
Τεχνικές που δοκιμάστηκαν

Αξίζει να επισημάνουμε ότι το βάθος της αναδρομής που επισημαίνεται σε κάθε μία τεχνική δύναται να είναι διαφορετικό από την επεξεργαστική ισχύ του υπολογιστή και για αυτό οι τιμές που παρουσιάζονται είναι ενδεικτικές. Μπορεί όμως να αλλάξει πολύ εύκολα μέσω της σταθεράς *MAX_DEPTH* στο **client.c** για να λειτουργεί απρόσκοπτα σε όλα τα συστήματα.

Minimax(depth limit)

Ο πρώτος αλγόριθμος που υλοποιήθηκε είναι ο **minimax**. Λόγω της απλότητας του αλλά και της έλλειψης ιδιαίτερης “εξυπνάδας” είχε όπως ήταν αναμενόμενο τις χειρότερες επιδόσεις αφού το βάθος της αναδρομής προκειμένου να λειτουργεί σε γρήγορο χρόνο ήταν **5-6.**

Ο minimax αποτελεί έναν backtracking που αποσκοπεί να εντοπίσει την βέλτιστη κίνηση για έναν παίκτη δεδομένου ότι ο αντίπαλο παίζει βέλτιστες κινήσεις. Αναζητεί όλες τις πιθανές κινήσεις εως ένα συγκεκριμένο βάθος, πραγματοποιεί εκτιμήσεις όλων των θέσεων, και αξιοποιεί αυτές τις εκτιμήσεις για να εντοπίσει το τελικό σκόρ στην ρίζα του δέντρου. Η αρχή του minimax είναι η ελαχιστοποίηση του μέγιστου του αντιπάλου, μεγιστοποιώντας το ελάχιστο που μπορούμε να φτάσουμε. Παρακάτω φαίνεται ενδεικτικά ένα δένδρο με την υλοποίηση minimax



A minimax tree example

Alpha-Beta Pruning

Ο αλγόριθμος αποτελεί μια βελτίωση του minimax που αναλύθηκε παραπάνω. Βασίζεται στην ίδια λογική με το αναδρομικό δέντρο αναζήτησης και εναλλαγή των παικτών με max(δικός μας) και min(αντίπαλος). Η διαφορά σε αυτή την περίπτωση είναι ότι ο αλγόριθμος minimax εξετάσει όλα τα φύλλα, ενώ ο alpha beta αποκόπτει όποια φύλλα δεν μπορούν να επηρεάσουν το αποτέλεσμα. Για αυτό τον λόγο εισάγονται δύο μεταβλητές alpha και beta, που αντιπροσωπεύουν το minimum σκορ του maximizer και το maximum score του minimizer αντίστοιχα. Όποτε λοιπόν το

maximum score του minimizer(beta παίκτης) βεβαιωθεί ότι γίνει μικρότερο από το minimum score του maximizer, ο maximizer δεν χρειάζεται να εξετάσει παραπάνω απογόνους του κόμβου αυτού μιας και δεν πρόκειται να επιλεγεί ποτέ. Έτσι το παράθυρο αναζήτησης μικραίνει με αρκετούς κόμβους να κλαδεύονται και συνεπώς αυξάνεται το βάθος της αναδρομής μας. Παράδειγμα φαίνεται στην παρακάτω εικόνα

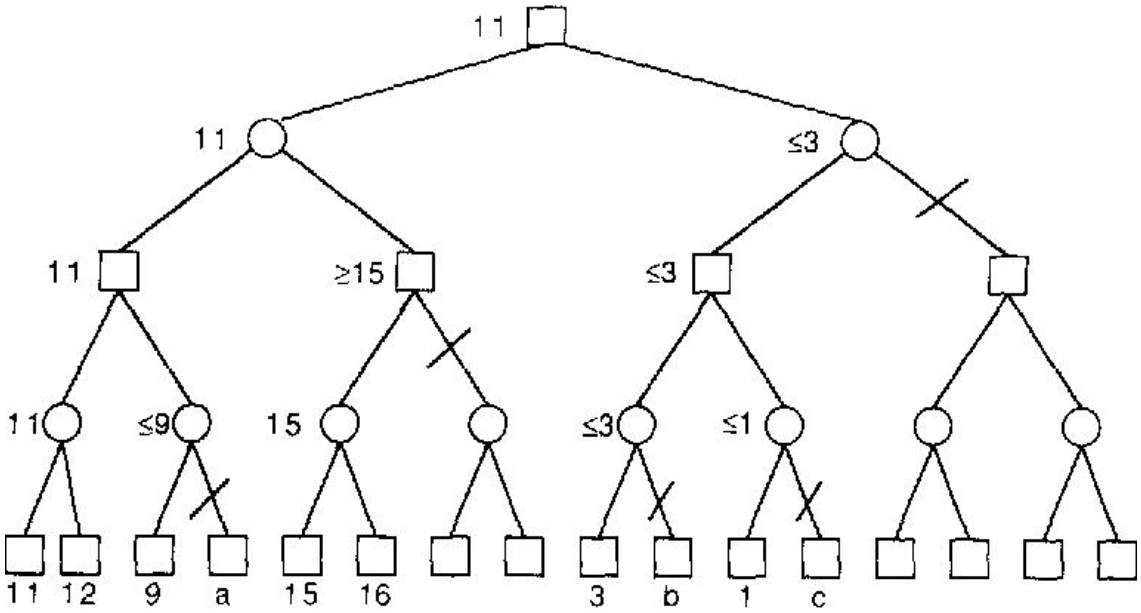


Fig. 1. Two-player alpha-beta pruning.

Με την συγκεκριμένη υλοποίηση αυξήθηκε το βάθος της αναδρομής στο **8**. Αξίζει να σημειωθεί ότι η υλοποίηση μόνη της έφτανε και στο **9** με σχετικά ικανοποιητικά αποτελέσματα αλλά με την **quiescence search** που προστέθηκε στην συνέχεια σε λογικά πλαίσια χρόνου εκτελείται ο αλγόριθμος για βάθος 8.

Να σημειωθεί ότι ελέγχθηκε ότι οι δύο πρώτοι αλγόριθμοι βγάζουν ακριβώς τα ίδια αποτελέσματα με manual τρόπο εκτελώντας τους αλγόριθμους για ίδιο βάθος και με τις ίδιες κινήσεις σαν αντίπαλος.

Quiescence search

Η συγκεκριμένη τεχνική δεν προστέθηκε για την αύξηση του βάθους της αναδρομής(όπως αναφέρθηκε μάλιστα χειροτέρεψε λίγο την εκτέλεση) αλλά βελτιώνει τις επιλογές του πράκτορα βλέποντας πιο σωστά κινήσεις. Μέχρι στιγμής όλες οι υλοποιήσεις που χρησιμοποιούμε πραγματοποιούν αναζήτηση σε ένα fixed

depth με αποτέλεσμα πιθανές απειλές ή ευκαιρίες ακριβώς μετά την αναζήτηση δεν ανιχνεύονται(γνωστό και ως horizon effect). Η quiescence αναζήτηση αυτό που κάνει είναι ότι δεν σταματά να ερευνά κινήσει οι οποίες δεν είναι “ήσυχες”. Το ποιες καταστάσεις θεωρούνται ήσυχες είναι στην κρίση του σχεδιαστή και ανάλογα με το παιχνίδι που υλοποιείται υπάρχουν διαφορετικές παραλλαγές. **Στην περίπτωσή μας, θεωρούμε ήσυχη την κατάσταση όπου δεν υπάρχουν άλλες πιθανές αιχμαλωτίσεις στο board για τον παίκτη μας**. Έγινε προσπάθεια ενσωμάτωσης αντίστοιχης κατάστασης και για τον αντίπαλο(με αρνητικό βάρος αυτές οι κινήσεις) αλλά πέφτει κατακόρυφα η απόδοση της αναζήτησης μας οπότε και δεν χρησιμοποιήθηκε.

Ακόμα οι συγκεκριμένη τεχνική, συνοδεύεται και με κάποιο είδος pruning συνήθως για να αντισταθμίζεται το επιπλέον βάθος αναδρομής και να μην επηρεάζεται η απόδοση. Εξετάστηκαν και υλοποιήθηκαν οι μέθοδοι **Delta Pruning** και **Null move pruning** αλλά σύντομα έγινε αντιληπτό ότι λόγω της απλοποίησης του παιχνιδιού (δεν έχουμε κίνηση προς τα πίσω) και επειδή οι αιχμαλωτίσεις είναι υποχρεωτικές δεν είχαν κάποια ιδιαίτερη αξία οι μέθοδοι αυτοί και παραλήφθηκαν απο την τελική υλοποίηση

MTD(F)

Ο συγκεκριμένος αλγόριθμος αποτελεί μια βελτίωση του αλγόριθμο alpha beta, όπου πάντα πραγματοποιεί αναζήτηση με minimal windows. Η αναζήτηση λειτουργεί προσεγγίζοντας την minimax τιμή. Στην πιο απλή μορφή της υλοποίησης, όπου και υλοποιήσαμε χρησιμοποιείται η απλή alpha beta συνάρτηση μας η οποία επιστρέφει ένα bound της minimax τιμής. Τα όρια αποθηκεύονται αντίστοιχα σε *upperbound* και *lowerbound* σχηματίζοντας ένα διάστημα γύρω από την πραγματική minimax τιμή. Όταν τόσο το πάνω όσο και το κάτω όριο συναντηθούν η minimax τιμή βρέθηκε. Επιπρόσθετα σημαντικό για την απόδοση του αλγορίθμου είναι να είναι καλή η πρώτη εκτίμηση της τιμής minimax.

Δυστυχώς, η υλοποίηση μας όχι μόνο δεν βελτιώνει την απόδοση αλλά την ρίχνει και επιβαιβεώθηκε και στην πράξη ότι η συγκεκριμένη μέθοδος λειτουργεί αποδοτικά μόνο με καλο move ordering και transposition tables όπου και γίνεται πιο επιθετικό pruning. Το βάθος που λειτουργεί ο αλγόριθμος είναι **7**.

NegaScout

Η μέθοδος Negascout αποτελεί παραλλαγή της μεθόδου alpha beta με χρήση minimal windows. Η βασική ιδέα πίσω από τον συγκεκριμένο αλγόριθμο είναι ότι οι περισσότερες κινήσεις μετά την πρώτη θα αποκοπούν οπότε ο υπολογισμός επακριβώς των τιμών τους δεν είναι χρήσιμος και αντί για αυτό προσπαθεί να δείξει ότι είναι κατώτερες μέσω αναζήτησης ενός minimal alpha-beta παραθύρου πρώτα. Για τα υποδέντρα λοιπόν που δεν μπορούν να βελτιώσουν την τιμή που έχει υπολογιστεί προηγουμένως ο NegaScout είναι ανώτερος του alpha beta λόγω του μικρότερου παραθύρου. Σε περιπτώσεις όμως που η κίνηση που εξετάζεται είναι καλύτερη επιλογή, το συγκεκριμένο υποδέντρο πρέπει να επανεπισκεφθεί εκ νέου, για να υπολογίσει ακριβώς την minimax τιμή.

Και σε αυτή την περίπτωση όπως καταλαβαίνουμε το move ordering παίζει καθοριστικό ρόλο αφού οι επανεπισκέψεις μπορούν να μειωθούν σε σημαντικό βαθμό αν οι κινήσεις είναι sorted με βάση την καλύτερη. Και για αυτή την υλοποίηση τα transposition tables θα βοηθούσουν σημαντικά και θα επιτάχυναν την διαδικασία. Σε αντίθεση όμως με την προηγούμενη μέθοδο είδαμε βελτίωση σε αυτή την περίπτωση με το βάθος της αναδρομής να φτάνει στο **10-11.**

Iterative Deepening

Η συγκεκριμένη υλοποίηση αποτελεί ακόμα μια λύση στο horizon effect όπως είχε αναλυθεί και στην Quiescence Search παραπάνω. Σε αυτή την περίπτωση το concept είναι ιδιαίτερα απλό αφού έχουμε ένα χρονόμετρο και αναζητούμε μέχρι να τελειώσει ο χρόνος μας χωρίς να αναζητούμε σε κάθε επανάληψη μέχρι ένα fixed depth. Θεωρείται επίσης ότι η συγκεκριμένη τακτική μπορεί να βοηθήσει στο move ordering που αποτελεί σημαντικό ζητούμενο, εφόσον περνάμε τα αποτελέσματα των προηγούμενων επαναλήψεων στην συνέχεια. Τα παραπάνω τα επαληθεύσαμε αφού σε συνδυασμό με την NegaScout αναζήτηση το βάθος της αναδρομής φτάνει στο 12 σε κάποιες περιπτώσεις εκτέλεσης.

ΣΗΜΑΝΤΙΚΟ: Λίγο πριν την παράδοση της εργασίας παρατηρήθηκε ένα μικρό bug μέσω του valgrind στην Negascout αναζήτηση. Προκειμένου να μην διακινδυνεύσουμε να "σκάσει" το πρόγραμμα κατά την πραγματοποίηση του τουρνουά αφήσαμε την επόμενη καλύτερη επιλογή μας δηλαδη alpha beta.

Evaluation Function

Η evaluation function αποτελεί ίσως τον πιο καθοριστικό παράγοντα στην τελική απόδοση του πράκτορα, αφού βάσει της εκτίμησης της επιλέγονται οι κινήσει και χτίζεται το δέντρο αναδρομής. Όσο πιο ακριβής η συνάρτηση εκτίμησης τόσο το καλύτερα, αλλά είναι σημαντικό να έχουμε στο μυαλό μας τον παράγοντα χρόνο και τελικά μας ενδιαφέρει να προκύψει μια αποδοτική και γρήγορη συνάρτηση. Η συνάρτηση μας λαμβάνει υπόψιν 3 παράγοντες: ***αριθμό πιονιών, heuristic value ανάλογα με την κάθε θέση στο board, αν στα πιονια του κάθε παίκτη υπάρχει προστασία***. Ο δικός μας παίκτης έχει θετική βαρύτητα στην τελική τιμή ενώ ο αντίπαλος έχει αρνητική.

Κάθε πιόνι έχει βαρύτητα **80**, στον δικό μας παίκτη οι επικαλύψεις των πιονιών έχουν (θετική)βαρύτητα **100** ενώ για τον αντίπαλο (αρνητική)βαρύτητα **150**. Τέλος στην κάθε θέση του board στον πίνακα **tableHeuristics** έχει ανατεθεί τιμή ανάλογα με το πως αξιολογούμε την κάθε θέση(πχ απο τις διαθέσιμες θέσεις προτιμούμε θέσεις στα πλαϊνά που το πιόνι μας δεν μπορεί να φαγωθεί)

Άλλες αλλαγές στον κώδικα

Μετά την εκπόνηση όλων αυτών των τεχνικών, αφού επιβεβαιώθηκε η ορθότητα τους ,αξιοποιήθηκαν κάποιες τεχνικές από άλλα μαθήματα για την βελτίωση της απόδοση της αναζήτησης. Αρχικά πραγματοποιήθηκαν αλλαγές σε όλους τους τύπους μεταβλητών προκειμένου να μην δεσμεύεται άχρηστος χώρος και να πραγματοποιούνται περισσότεροι υπολογισμοί. Σε κάποια σημεία έγινε ακόμα και ανάθεση μεταβλητών με συγκεκριμένο μέγεθος bit(βλέπε αρχείο client.h). Απο κει και πέρα αξιοποιήθηκε η τεχνική του loop unrolling σε διάφορα σημεία όπως γίνεται αντιληπτό από ανάγνωση του κώδικα που βελτιώνει σε ένα βαθμό την ταχύτητα των

υπολογισμών. Όλες αυτές οι αλλαγές βελτίωσαν σε μικρό βαθμό την απόδοση του κώδικα με τις μεθόδους να δουλεύουν καλύτερα όπως παρατηρήθηκε στις ακραίες περιπτώσεις(χρονικά)

Future Work

Περιοριστικό παράγοντα στο πρότζεκτ έπαιξε εξ αρχής η γλώσσα υλοποίησης C, η οποία παρουσιάζει αρκετά προβλήματα ειδικά στις δυναμικές δομές αποθήκευσης πράγμα που περιόρισε τον χρόνο και τις διαφορετικές τεχνικές που αξιοποιήθηκαν. Η επιλογή της C καθώς ο server και ο client είχε δοθεί έτοιμος σε C και δεν υπήρξε χρόνος για τόσο μεγάλες τροποποιήσεις.

Αναφορικά με τις τεχνικές, θα έκανε μεγάλη διαφορά σίγουρα κάποια τεχνική όπως transposition tables που θα βελτίωνε σε μεγάλο βαθμό το move ordering γλιτώνοντας και πολλαπλές ίδιες αναζητήσεις που γίνεται στις τρέχουσες αναζητήσεις. Ακόμα, ιδιαίτερα ελπιδοφόρα έδειχνε η μέθοδος για χρήση bitboard πράγμα όμως που θα απαιτούσε την ολική διαφοροποίηση του μηχανισμού που δόθηκε απο την εκφώνηση. Τέλος ορισμένες τεχνικες που θα βελτίωναν σίγουρα την απόδοση ήταν και χρησιμοποιούνται απο το Chinook(τέλειο παιχνίδι) είναι η τεχνική των opening book και των endgame databases που όμως δεν γίνεται να χρησιμοποιήσουμε στην υλοποίηση μας καθώς και μια πιο βέλτιστη evaluation function.

Βιβλιογραφία

Διαλέξεις Μαθήματος

http://www.fierz.ch/strategy2.htm#history

https://webdocs.cs.ualberta.ca/~mmueller/courses/2014-AAAI-games-tutorial/slides/AAAI-14-Tutorial-Games-3-AlphaBeta.pdf

https://homepages.cwi.nl/~paulk/theses/Carolus.pdf

https://web.archive.org/web/20180714104141/https://chessprogramming.wikispaces.com/Search

https://www.cs.unm.edu/~aaron/downloads/qian_search.pdf

https://en.wikipedia.org/wiki/Minimax

https://arxiv.org/pdf/1404.1511.pdf