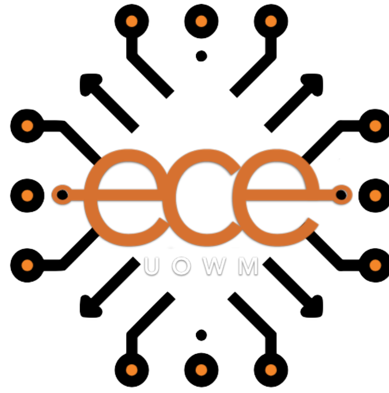


Πανεπιστήμιο Δυτικής Μακεδονίας

Τμήμα Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών

Αλγόριθμοι για την Επίλυση Συνδυαστικών Παιγνίων

Ιωάννης Συγκούνας (ΑΜ: 1556)

Επιβλέπων Καθηγητής: Νικόλαος Πλόσκας

Εργαστήριο Ευφυών Συστημάτων & Βελτιστοποίησης

10 Οκτωβρίου 2023

Περίληψη

Το θέμα στο οποίο επικεντρώνεται η συγκεκριμένη διπλωματική είναι τα συνδυαστικά παιχνίδια, μία υποκατηγορία των συνδυαστικών προβλημάτων βελτιστοποίησης. Ο καθορισμός της ιδανικής κίνησης με βάση τη διαθέσιμη πληροφορία που έχει τη δεδομένη στιγμή ο παίκτης είναι το κύριο πρόβλημα με το οποίο ασχολείται ο κλάδος. Με βάση αυτό λοιπόν, τα προβλήματα που μελετούνται είναι η αήττητη υλοποίηση ενός Min-Max πράκτορα για το παιχνίδι της τρίλιζας, η σύγκριση των Min-Max και Alpha-Beta Pruning στο παιχνίδι Connect-4, ο καθορισμός του ιδανικότερου αλγορίθμου μεταξύ των Breadth-First Search (BFS), Depth-First Search (DFS), A* (με Manhattan Distance ως Heuristic), A* (με Manhattan Distance ως Heuristic αλλά και Reversal Penalty), Branch-And-Bound Search (B&B) και Iterative-Deepening Search (IDS) για την επίλυση του 8-Puzzle και ομοίως ο καθορισμός του ιδανικότερου αλγορίθμου μεταξύ των προαναφερόμενων για την εύρεση του συντομότερου μονοπατιού του ιππότη πάνω σε μία σκακιέρα με $N \times N$ διαστάσεις.

Λέξεις κλειδιά: Παιχνίδια συνδυαστικής βελτιστοποίησης, Min-Max, Alpha-Beta Pruning, BFS, DFS, A*, Branch & Bound, IDS.

Abstract

The topic on which this thesis focuses is combinatorial games, a subclass of combinatorial optimization problems. Determining the ideal move based on the information available to the player at any given moment is the main problem this research field is dealing with. Based on this, the problems studied are, an unbeatable implementation of a Min-Max agent for Tic-Tac-Toe, the comparison between the algorithms Min-Max and Alpha-Beta Pruning in the game Connect-4, the determination of the most ideal algorithm among Breadth-First Search (BFS), Depth-First Search (DFS), A* (with Manhattan Distance as Heuristic), A* (with Manhattan Distance as Heuristic but also Reversal Penalty), Branch-And-Bound Search (B&B) and Iterative-Deepening Search (IDS) for solving the 8-Puzzle and similarly the determination of the most ideal algorithm among the aforementioned for finding the shortest path of the knight pawn on a chessboard with N × N dimensions.

Keywords: Combinatorial games, Min-Max, Alpha-Beta Pruning, BFS, DFS, A*, Branch & Bound, IDS.

Δήλωση Πνευματικών Δικαιωμάτων

Δήλωση Πνευματικών Δικαιωμάτων Δηλώνω ρητά ότι, σύμφωνα με το άρθρο 8 του Ν. 1599/1986 και τα άρθρα 2,4,6 παρ. 3 του Ν. 1256/1982, η παρούσα Διπλωματική Εργασία με τίτλο "Αλγόριθμοι για την Επίλυση Συνδυαστικών Παιγνίων" καθώς και τα ηλεκτρονικά αρχεία και πηγαίοι κώδικες που αναπτύχθηκαν ή τροποποιήθηκαν στα πλαίσια αυτής της εργασίας και αναφέρονται ρητώς μέσα στο κείμενο που συνοδεύουν, και η οποία έχει εκπονηθεί στο Τμήμα Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών του Πανεπιστημίου Δυτικής Μακεδονίας, υπό την επίβλεψη του μέλους του Τμήματος κ. Νικόλαου Πλόσκα αποτελεί αποκλειστικά προϊόν προσωπικής εργασίας και δεν προσβάλλει κάθε μορφής πνευματικά δικαιώματα τρίτων και δεν είναι προϊόν μερικής ή ολικής αντιγραφής, οι πηγές δε που χρησιμοποιήθηκαν περιορίζονται στις βιβλιογραφικές αναφορές και μόνον. Τα σημεία όπου έχω χρησιμοποιήσει ιδέες, κείμενο, αρχεία ή / και πηγές άλλων συγγραφέων, αναφέρονται ευδιάκριτα στο κείμενο με την κατάλληλη παραπομπή και η σχετική αναφορά περιλαμβάνεται στο τμήμα των βιβλιογραφικών αναφορών με πλήρη περιγραφή.

Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας εργασίας, εξ ολοκλήρου ή τμήματος αυτής, για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα. Ερωτήματα που αφορούν τη χρήση της εργασίας για κερδοσκοπικό σκοπό πρέπει να απευθύνονται προς τον συγγραφέα. Οι απόψεις και τα συμπεράσματα που περιέχονται σε αυτό το έγγραφο εκφράζουν τον συγγραφέα και μόνο.

Copyright (C) Ιωάννης Συγκούνας & Νικόλαος Πλόσκας, 2023, Κοζάνη

Υπογραφή Φοιτητή

Περιεχόμενα

1 Εισαγωγή **9**

1.1 Ορισμός του προβλήματος 9

1.2 Κίνητρα και Στόχοι Υλοποίησης 10

1.3 Διάρθρωση κειμένου 10

2 Βιβλιογραφική ανασκόπηση **11**

3 Υλοποίηση **26**

3.1 Τρίλιζα 26

3.1.1 Min-Max 26

3.2 Connect-4 28

3.2.1 Aplha-Beta Pruning 28

3.3 8-Puzzle και Knight Problem 30

3.3.1 Breadth First Search - BFS 30

3.3.2 Depth First Search - DFS 31

3.3.3 A* Search (Manhattan distance) 33

3.3.4 A* Search (Manhattan distance and reversal penalty) 35

3.3.5 Branch and Bound - B&B 38

3.3.6 Iterative Deepening Search - IDS 41

4 Υπολογιστική μελέτη **44**

4.1 Μετρικές και Προβλήματα 44

4.2 Αποτελέσματα πειραμάτων 45

4.2.1 Τρίλιζα 45

4.2.2 Connect-4 45

4.2.3 8-Puzzle 49

4.2.4 Knight Problem 52

4.3 Συμπεράσματα πειραμάτων 54

5 Συμπεράσματα **56**

Κατάλογος σχημάτων

2.1 Καλές στρατηγικές για τρίλιζα [2] 14

3.1 Αλγοριθμικό δέντρο ενός Min-Max αλγόριθμου 28
3.2 Αλγοριθμικό δέντρο ενός Alpha-Beta Pruning αλγόριθμου 30
3.3 Αλγοριθμικό δέντρο ενός Breadth First Search αλγόριθμου 31
3.4 Αλγοριθμικό δέντρο ενός Depth First Search αλγόριθμου 32
3.5 Αλγοριθμικό δέντρο ενός A* Search (Manhattan distance) αλγόριθμου 35
3.6 Αλγοριθμικό δέντρο ενός A* Search (Manhattan distance and reversal penalty) αλγόριθμου 38
3.7 Αλγοριθμικό δέντρο ενός Branch and Bound αλγόριθμου 40
3.8 Αλγοριθμικό δέντρο ενός Iterative Deepening Search αλγόριθμου . . . 43

4.1 Αναλογία νικών μεταξύ παικτών για Alpha-Beta ενάντια Alpha-Beta . 46
4.2 Αναλογία νικών μεταξύ παικτών για Min-Max ενάντια Min-Max . . . 47
4.3 Αναλογία νικών μεταξύ παικτών για Min-Max ενάντια Alpha-Beta . . 48
4.4 Αναλογία νικών μεταξύ παικτών για Alpha-Beta ενάντια Min-Max . . 49
4.5 Μέσος όρος χρόνου εκτέλεσης από 100 μετρήσεις 50
4.6 Μέσος όρος αριθμού κόμβων που επεκτάθηκαν από 100 μετρήσεις . . 51
4.7 Μέσος όρος χρόνου εκτέλεσης από 100 μετρήσεις 53
4.8 Μέσος όρος αριθμού κόμβων που επεκτάθηκαν από 100 μετρήσεις . . 54

Κατάλογος αλγορίθμων

1 Min-Max 27
2 Alpha-Beta Pruning 29
3 Breadth First Search - BFS 31
4 Depth First Search - DFS 32
5 A* Search (Manhattan distance) - Part 1 33
6 A* Search (Manhattan distance) - Part 2 34
7 A* Search (Manhattan distance and reversal penalty) - Part 1 36
8 A* Search (Manhattan distance and reversal penalty) - Part 2 37
9 Branch and Bound - B&B - Part 1 39
10 Branch and Bound - B&B - Part 2 40
11 Iterative Deepening Search - IDS 42

Κατάλογος πινάκων

4.1 Alpha-Beta ενάντια Alpha-Beta 46
4.2 Min-Max ενάντια Min-Max 46
4.3 Min-Max ενάντια Alpha-Beta 47
4.4 Alpha-Beta ενάντια Min-Max 48
4.5 Μέσοι όροι του 8-Puzzle. 50
4.6 Μέσοι όροι του Knight Problem. 53

Κεφάλαιο 1

Εισαγωγή

1.1 Ορισμός του προβλήματος

Τα συνδυαστικά προβλήματα περιλαμβάνουν εργασία με διακριτές μεταβλητές για την ικανοποίηση συγκεκριμένων συνθηκών. Όπως αναφέρει ο Stutzle [18] τα προβλήματα μπορούν να χωριστούν σε δύο κύριες κατηγορίες: προβλήματα βελτιστοποίησης και ικανοποίησης. Στα προβλήματα βελτιστοποίησης, ο στόχος είναι να βρεθεί η καλύτερη διάταξη, ομαδοποίηση, σειρά ή επιλογή διακριτών αντικειμένων, συχνά με πεπερασμένο αριθμό επιλογών. Στα προβλήματα ικανοποίησης, ο στόχος είναι να βρεθεί μια λύση που να πληροί δεδομένους περιορισμούς. Ένα κλασικό παράδειγμα είναι το πρόβλημα ικανοποίησης περιορισμών, όπου προσδιορίζεται εάν είναι δυνατό να βρεθούν τιμές για μεταβλητές με τρόπο που να ικανοποιεί ένα σύνολο περιορισμών. Η ιδανική προσέγγιση των προβλημάτων συνδυαστικής βελτιστοποίησης, στα οποία επικεντρώνεται η συγκεκριμένη διπλωματική, είναι να εξεταστούν όλες οι πιθανές λύσεις, κάτι που είναι όμως συχνά ανέφικτο για μεγάλους χώρους αναζήτησης. Αυτό, λοιπόν, είναι το σημείο που ξεκινούν οι διαφοροποιήσεις τόσο σε τρόπους προσέγγισης των προβλημάτων όσο και σε κατηγορίες και είδη αλγορίθμων.

Οι Alazzam et al. [1] επιλέγουν την ανάπτυξη και τη μελέτη μετά-ευρετικών αλγορίθμων για τη λύση του προβλήματος. Ο Johnson [12] αλλάζει την προσέγγιση και ερευνά πόσο κακή μπορεί να είναι η απόδοση ενός αλγορίθμου συγκριτικά με την καλύτερη δυνατή λύση για να δει πως αυτό επηρεάζεται όσο μεγαλώνει το πρόβλημα. Οι Hegerty et al. [9] επιλέγουν να συγκρίνουν δύο τεχνικές βελτιστοποίησης, τη διαφορική εξέλιξη και τους γενετικούς αλγόριθμους, για την επίλυση δύσκολων

συνδυαστικών προβλημάτων όπως τα προβλήματα N-Queens και πλανόδιου πωλητή, τα οποία είναι γνωστό ότι είναι πολύ πολύπλοκα.

1.2 Κίνητρα και Στόχοι Υλοποίησης

Η συγκεκριμένη διπλωματική ασχολείται με την πρώτη κατηγορία συνδυαστικών προβλημάτων, τα προβλήματα βελτιστοποίησης, και μάλιστα με μία συγκεκριμένη υποκατηγορία αυτών, τα συνδυαστικά παιχνίδια. Σκοπός είναι η υλοποίηση αλγορίθμων πάνω στα περιβάλλοντα της τρίλιζας, του Connect-4, του 8-Puzzle και της $N \times N$ σκακιέρας με σκοπό την καταγραφή αποτελεσμάτων και τη σύγκριση αυτών.

1.3 Διάρθρωση κειμένου

Τα υπόλοιπα κεφάλαια οργανώνονται ως εξής. Στο Κεφάλαιο 2 παρουσιάζεται το επιστημονικό πεδίο ενασχόλησης της διπλωματικής καθώς και με τι έχουν ασχοληθεί άλλοι ερευνητές σε αυτό. Ακολουθεί η υλοποίηση στο Κεφάλαιο 3, όπου παρουσιάζονται οι αλγόριθμοι που αξιοποιήθηκαν για να λυθούν τα προβλήματα, ο ψευδοκώδικας τους, άλλα και η σχηματική αναπαράσταση της λειτουργίας τους. Συνέχεια έχουν οι μετρικές και τα προβλήματα στο Κεφάλαιο 4, όπου παρουσιάζονται τα ίδια τα προβλήματα που εξετάζονται σε αυτήν τη διπλωματική, το περιβάλλον στο οποίο υλοποιήθηκαν καθώς και οι μετρήσεις που έγιναν, από τις οποίες προκύπτουν οι πίνακες και τα διαγράμματα με όλα τα στοιχεία που οδηγούν στα συμπεράσματα της υπολογιστικής μελέτης. Τα συμπεράσματα της εργασίας παρατίθενται στο Κεφάλαιο 5.

Κεφάλαιο 2

Βιβλιογραφική ανασκόπηση

Η οικογένεια των συνδυαστικών παιχνιδιών αποτελείται από παζλ ενός παίκτη ή από παιχνίδια δύο παικτών όπου υπάρχει απόλυτη γνώση για την κατάσταση του εκάστοτε ”ταμπλό”, δεν υπάρχει ο παράγοντας της τύχης (π.χ. ζάρια) και στην περίπτωση της δεύτερης ομάδας το αποτέλεσμα μπορεί να είναι μία νίκη ή μία ισοπαλία. Τέτοια παιχνίδια είναι τόσο το 8-puzzle problem, ένα παζλ που αποτελείται από 8 αριθμούς, ένα κενό και σκοπός είναι να βρεθεί ο σωστός συνδυασμός κινήσεων έτσι ώστε να τοποθετηθούν οι αριθμοί και το κενό στη σωστή σειρά, όσο και το Connect-4, όπου σκοπός του κάθε παίχτη είναι να ”συνδέσει” τέσσερις από τους έγχρωμους δίσκους που του αντιστοιχούν κάθετα, οριζόντια ή διαγώνια. Ακόμα πιο γνωστά συνδυαστικά παιχνίδια είναι το σκάκι (chess), τα πούλια (checkers), το Go και φυσικά η τρίλιζα (tic-tac-toe) [6]. Παραδόξως, πολλά φαινομενικά απλά παζλ και παιχνίδια είναι επίσης δύσκολα και υπάρχουν ακόμα και περιπτώσεις που ορισμένα προβλήματα σε κάποια συνδυαστικά παιχνίδια είναι αδύνατα να λυθούν [7]. Το κύριο πρόβλημα των παιχνιδιών συνδυαστικής, λοιπόν, είναι ο καθορισμός της ιδανικής κίνησης [5] με βάση τη διαθέσιμη πληροφορία που έχει ο παίκτης. Για αυτόν ακριβώς τον λόγο τα παιχνίδια αυτά έχουν κεντρίσει το ενδιαφέρον σε ερευνητές από όλο τον κόσμο, οι οποίοι εφαρμόζουν γνωστούς αλγορίθμους για να απαντήσουν μία πληθώρα ερωτημάτων, είτε αυτά έχουν να κάνουν με το αν κάποιος αλγόριθμος μπορεί να λύσει ή παίξει ένα τέτοιο παιχνίδι, είτε για το αν κάποιος αλγόριθμος μπορεί να αποτελέσει καλύτερη επιλογή σε σύγκριση με κάποιον άλλον. Εφαρμόζονται λοιπόν διάφορες τεχνικές βελτίωσης πάνω σε γνωστούς αλγορίθμους παρέχοντας έτσι καλύτερα αποτελέσματα και ανακαλύψεις που γεννούν νέες τεχνικές και νέες, πιο στοχευμένες, υλοποιήσεις αλγορίθμων.

Ο Kunkle [13] παρουσιάζει τη λύση ενός 8-puzzle με την αξιοποίηση του αλγορίθμου A*. Στην αρχή γίνεται η επίλυση του παζλ με τον BFS (Breadth-First Search ή Αναζήτηση πρώτα σε πλάτος) που είναι ένας αλγόριθμος τυφλής αναζήτησης και χρησιμοποιείται ως μέτρο σύγκρισης για τις δύο υλοποιήσεις του A* που ακολουθούν. Ο A* είναι ένας αλγόριθμος ενημερωμένης αναζήτησης ο οποίος αξιοποιεί μία ευρετική διαδικασία με σκοπό να βρεθεί η βέλτιστη λύση. Στη συγκεκριμένη υλοποίηση γίνεται χρήση δύο διαφορετικών ευρετικών διαδικασιών, της απόστασης Manhattan και της απόστασης Manhattan με Reversal Penalty. Η απόσταση Manhattan υπολογίζει την απόσταση της τωρινής θέσης από αυτήν του στόχου, περιορίζοντας έτσι την αναζήτηση, με την προσθήκη του Reversal Penalty προστίθεται επίσης ένα κόστος κάθε φορά που γίνεται αλλαγή της κατεύθυνσης περιορίζοντας την αναζήτηση ακόμα πιο πολύ και δίνοντας έτσι τη βέλτιστη λύση ακόμα πιο σύντομα. Αυτό παρουσιάζεται στις δοκιμές που έγιναν, όπου με τη χρήση της απόστασης Manhattan αναπτύσσονται 25 φορές λιγότεροι κόμβοι στο δένδρο αναζήτησης και εφαρμόζοντας και το Reversal Penalty στην απόσταση Manhattan ο αλγόριθμος λαμβάνει περίπου 16% περισσότερη πληροφορία.

Η Iordan [11] υλοποιεί τη λύση ενός 8-puzzle με την αξιοποίηση του αλγορίθμου A* αλλά αυτήν τη φορά χρησιμοποιώντας και μία βελτιωμένη ευρετική διαδικασία. Οι ευρετικές διαδικασίες που χρησιμοποιούνται ως μέτρο σύγκρισης στη συγκεκριμένη υλοποίηση είναι δύο. Η πρώτη είναι η απόσταση Manhattan και η λειτουργία της αναφέρεται σε άλλο άρθρο παραπάνω. Η δεύτερη είναι η απόσταση Hamming η οποία μετράει πόσοι αριθμοί δεν είναι στη θέση που θα έπρεπε να είναι. Η ευρετική διαδικασία στην οποία επικεντρώνεται το άρθρο είναι η απόσταση Chebyshev η οποία δουλεύει όπως και η απόσταση Manhattan, αλλά βρίσκει και κρατάει τη μέγιστη απόσταση μεταξύ της κάθετης και οριζόντιας απόστασης της τωρινής θέσης σε σύγκριση με αυτήν του στόχου. Μετά από τις δοκιμές και τις συγκρίσεις που γίνονται μεταξύ των ευρετικών διαδικασιών βγαίνει ένα πόρισμα πως αξιοποιώντας τη διπλάσια απόσταση Chebyshev ως ευρετική διαδικασία έχουμε ένα πιο ικανοποιητικό αποτέλεσμα από τις άλλες δύο υλοποιήσεις.

Ο Reinefeld [16] αναφέρει πως το 8-puzzle είναι το μεγαλύτερο σε διαστάσεις παζλ (3×3) της κατηγορίας του που μπορεί να λυθεί πλήρως. Στο συγκεκριμένο παζλ εφαρμόζεται αρχικά ο αλγόριθμος IDA* με την απόσταση Manhattan ως ευ-

ρετική διαδικασία. Ο IDA* είναι μια επέκταση του A* που μοιάζει με αναζήτηση σε βάθος (DFS). Η κυρίαρχη ιδέα του IDA* είναι η αντικατάσταση του σταθερού ορίου βάθους του συμβατικού DFS με έναν αυξανόμενο περιορισμό βάθους κατά την αναζήτηση. Έτσι ο IDA* μπορεί να πραγματοποιήσει μία αναζήτηση πρώτα σε βάθος ενώ παράλληλα να αποκλείει κόμβους που δε θα δώσουν καλύτερη λύση από αυτή που έχει ήδη ανακαλυφθεί. Ο κύριος σκοπός του συγκεκριμένου άρθρου όμως είναι να δούμε πως το Node Ordering, δηλαδή η σειρά με την οποία επεκτείνονται οι κόμβοι, μπορεί να επηρεάσει το αποτέλεσμα της αναζήτησης. Τρέχοντας τον IDA* λοιπόν για όλους τους πιθανούς συνδυασμούς του 8-puzzle παρατηρήθηκε πως μία προκαθορισμένη σειρά επέκτασης (π.χ. πάνω, κάτω, δεξιά, αριστερά) δεν αποδίδει τόσο καλά και πως σε αυτή την περίπτωση η τυχαία επιλογή σε κάθε επέκταση είναι καλύτερη. Δοκιμάστηκαν επίσης οι ευρετικές διαδικασίες:

- Longest Path First, όπου η διάταξη των κόμβων καθορίζεται από το μήκος της μεγαλύτερης διαδρομής που τελειώνει στον εκάστοτε κόμβο.

- Steepest Ascent Hill-Climbing, όπου η διάταξη των κόμβων καθορίζεται από ποιος κόμβος έχει το μικρότερο κόστος με την ελπίδα ότι θα είναι και πιο κοντά στον στόχο.

- History Heuristic, όπου η διάταξη των κόμβων καθορίζεται αξιολογώντας προηγούμενες επεκτάσεις αποφεύγοντας έτσι τις επανεξετάσεις ορισμένων κόμβων.

- Transposition Table, όπου αποθηκεύει τη μορφή του ”ταμπλό” σε κάθε περίπτωση, το βάθος στο οποίο αναζητήθηκε και την καλύτερη κίνηση από εκεί. Παρ’ όλα αυτά το Node Ordering δεν παίζει σημαντικό ρόλο σε αυτήν την περίπτωση.

Προκύπτει λοιπόν, μέσα από όλες αυτές τις δοκιμές πως η καλύτερη επιλογή είναι το Longest Path First heuristic καθώς είναι το πιο αποτελεσματικό και με εύκολη υλοποίηση.

Οι Bhatt et al. [2] παρουσιάζουν την τρίλιζα (Tic-Tac-Toe), ένα από τα δημοφιλέστερα παιχνίδια συνδυαστικής. Η πλειοψηφία των παρτίδων καταλήγει σε ισοπαλία, επομένως το καλύτερο στο οποίο μπορεί να ελπίζει ένας παίχτης είναι να μη χάσει.

Σκοπός του άρθρου είναι να χρησιμοποιηθεί ένας κατάλληλα τροποποιημένος Γενετικός Αλγόριθμος (Customized Genetic Algorithm) με σκοπό να βρεθούν όσο το δυνατόν περισσότερες στρατηγικές ώστε να μη χάσει κάποιος σε μια παρτίδα τρίλιζας. Σε έναν τέτοιου τύπου γενετικό αλγόριθμο, δημιουργείται ένας πληθυσμός πιθανών λύσεων και κάθε λύση αναπαρίσταται ως μία σειρά από τιμές ή "γονίδια". Αυτά τα γονίδια στη συνέχεια μεταλλάσσονται και ανασυνδυάζονται για να παράγουν νέες λύσεις, οι οποίες αξιολογούνται σύμφωνα με μία συνάρτηση φυσικής κατάστασης που μετρά πόσο καλά λύνουν το πρόβλημα. Οι πιο κατάλληλες λύσεις επιλέγονται για να "επιβιώσουν" και να περάσουν τα γονίδιά τους στην επόμενη γενιά, ενώ οι πιο αδύναμες λύσεις απορρίπτονται. Αυτή η διαδικασία επαναλαμβάνεται για πολλές γενιές μέχρι να βρεθεί μια ικανοποιητική λύση. Αφού δημιουργήθηκε ένας τυχαίος αρχικός πληθυσμός λοιπόν πρέπει να γίνει μια αξιολόγηση των στρατηγικών που χρησιμοποιήθηκαν η οποία έχει την εξής συνάρτηση: $\text{Fitness} = \frac{\text{Number of games lost}}{\text{Number of games played}}$. Στη συνέχεια όμως, για να διατηρηθεί η ποικιλία των λύσεων χρησιμοποιείται ο τελεστής στοχαστικής ομοιόμορφης επιλογής (stochastic uniform selection - SUS) και η παραπάνω συνάρτηση παίρνει τη μορφή: $\text{Fitness} = \frac{1}{\text{m}}(1 - \frac{\text{Number of games lost}}{\text{Number of games played}})$, όπου m είναι ο αριθμός των μελών του πληθυσμού που έχουν την ίδια αναλογία ηττών. Ένας τυπικός τελεστής συγχώνευσης πολλαπλών σημείων (standard multi-point crossover operator) εφαρμόζεται για τη δημιουργία των γονιδίων και στα γονίδια με την ελάχιστη καταλληλότητα συνδέεται η πιθανότητα μετάλλαξης, ώστε ένας πληθυσμός που έχει χειρότερες λύσεις να έχει μεγαλύτερη πιθανότητα μετάλλαξης και το αντίστροφο. Ο ανασυνδυασμός γίνεται με έναν ιδιαίτερο τρόπο καθώς συνδυάζονται τρεις πληθυσμοί: ένας πριν την αξιολόγηση, ένας μετά τη συγχώνευση και ένας μετά τη μετάλλαξη. Κλείνοντας, λοιπόν, συμπεραίνεται πως οι δυο παίκτες χρησιμοποιούν τελείως ξεχωριστές στρατηγικές και εφαρμόζοντας το γενετικό αλγόριθμο ξεχωριστά στον καθένα ανακαλύφθηκαν $72,657$ στρατηγικές μεταξύ των οποίων ξεχωρίζουν τέσσερις.

Strategy	As first player		As second player		Win/ Draw
	Wins	Draws	Wins	Draws	
Mode	41	5	75	68	1.59
Mode-move	44	8	88	61	1.91
Best W-t-D	42	3	98	35	3.68
Mod. heuristic	18	1	68	21	3.91

Σχήμα 2.1: Καλές στρατηγικές για τρίλιζα [2]

Επίσης προκύπτει το πόρισμα πως ο πρώτος παίχτης έχει πλεονέκτημα καθώς είναι πιο εύκολο να ανακτηθεί μια στρατηγική που να αποτρέψει την ήττα του.

Ο Hochmuth [10] αναφέρεται σε μία καλή στρατηγική στο συνδυαστικό παιχνίδι της τρίλιζας που εγγυάται περισσότερο την αποφυγή της ήττας και όχι τη νίκη. Με αυτό ως βάση αξιοποιείται, σε αυτό το άρθρο, γενετικός αλγόριθμος με στόχο την εύρεση μίας στρατηγικής που ουσιαστικά δε θα χάνει ποτέ. Σε ένα "ταμπλό" (3×3) τρίλιζας μπορούν να υπάρξουν μέχρι και $362,880$ μοναδικές εκδοχές συνδυασμών εκ των οποίων όμως μια μεγάλη πληθώρα δεν είναι αποδεκτές, για παράδειγμα όλα τα κουτάκια να γεμίσουν με X. Επίσης υπάρχουν συνδυασμοί που είναι συμμετρικοί μεταξύ τους αν γυρίσει κατάλληλα το ταμπλό, έτσι ο αριθμός των μοναδικών συνδυασμών μειώνεται σημαντικά στους 827. Σύμφωνα με τον γενετικό αλγόριθμο, ο πίνακας χαρτογράφησης και το "γονιδίωμα" ενός "ατόμου" έχουν 827 μοναδικά "γονίδια", η ενέργεια που πρέπει να γίνει σε αυτήν την περίσταση ορίζει την αξία κάθε γονιδίου. Στη συνέχεια ακολουθεί η αξιολόγηση του κάθε γονιδίου ή στρατηγικής για εμάς, η οποία γίνεται με τον εξής τύπο: $\text{Fitness} = \frac{\text{Number of games played} - \text{Number of games lost}}{\text{Number of games played}}$. Τα γονίδια με την καλύτερη αξιολόγηση έχουν τη μεγαλύτερη πιθανότητα να μεταδώσουν την πληροφορία τους στην επομένη "γενιά". Σειρά τώρα έχουν η "αναπαραγωγή", η "μετάλλαξη" και η "διασταύρωση". Το στάδιο της αναπαραγωγής αντιγράφει ένα άτομο με πιθανότητα $p - replication$ στην επόμενη γενιά χωρίς τροποποίηση. Η μετάλλαξη μπορεί να συμβεί σε κάθε γονίδιο ενός ατόμου με πιθανότητα $p - mutate$ αλλάζοντας την αξία του γονιδίου με μια νέα τυχαία τιμή και στη συνέχεια τα γονίδια διασταυρώνονται με βάση μία πιθανότητας $p - crossover$ η οποία διαλέγει πότε θα λαμβάνει πληροφορίες από τον έναν "γονέα" και πότε από τον άλλον. Τρέχοντας λοιπόν τον αλγόριθμο βρέθηκε μια κατάλληλη στρατηγική για την αποφυγή της ήττας σε μόλις 373 γενιές, παράχθηκαν 500 στρατηγικές και καμία από αυτές δεν έχασε καμία παρτίδα.

Οι Hauptman και Sipper [8] αναφέρονται στην ανάπτυξη αλγορίθμων αναζήτησης για τη λύση του Mate-in-N προβλήματος σε μια σκακιέρα με βάση τον γενετικό προγραμματισμό. Σκοπός του Mate-in-N προβλήματος είναι να βρεθεί μία χαρακτηριστική κίνηση ή κίνηση κλειδί χάρις την οποία ο αντίπαλος να μη μπορεί να αποφύγει το Mate σε N ή λιγότερες κινήσεις (όχι μεικτές, μόνο του παίχτη που επιτίθεται). Για τη συγκεκριμένη υλοποίηση χρησιμοποιείται γενετικός προγραμ-

ματισμός τύπου Koza, δηλαδή μια τεχνική μηχανικής μάθησης που χρησιμοποιεί γενετικούς αλγόριθμους για τη μετάλλαξη προγραμμάτων που μπορούν να λύσουν ένα συγκεκριμένο πρόβλημα. Τα ''άτομα'' του ''γονιδιώματος'' σε αυτήν την περίπτωση είναι εκφράσεις LISP. Οι εκφράσεις LISP είναι ένας τρόπος αναπαράστασης προγραμμάτων και δεδομένων στη γλώσσα προγραμματισμού LISP. Το LISP σημαίνει ''LISt Processing'' και είναι μια οικογένεια γλωσσών προγραμματισμού που βασίζονται στην έννοια των συνδεδεμένων λιστών. Στη συγκεκριμένη περίπτωση οι LISP εκφράσεις είναι συναρτήσεις και τερματικά για την υλοποίηση των οποίων λήφθηκε υπόψιν η γνώμη πολλών υψηλόβαθμων παιχτών του σκακιού. Για την αξιολόγηση των ατόμων ή των αλγορίθμων αναζήτησης για εμάς αξιοποιήθηκαν 100 Mate-in-N προβλήματα και ανάλογα με το πόσο καλά καταφέρνει κάθε άτομο να λύσει έναν τυχαίο αριθμό από αυτά του αντιστοιχείται ένα σκορ. Το σκορ αυτό προκύπτει από τον εξής τύπο: $fitness = \sum_{i=1}^{s \cdot \text{Max}_N} \text{Correctness}_i \cdot 2^{N_i} \cdot \text{Boards}_i$. Όπου τα:

- i, N και s είναι η περίπτωση του προβλήματος, το βάθος και το μέγεθος του δείγματος, αντίστοιχα. Το Max_N είναι το μέγιστο βάθος.

- Correctness_i $[0, 1]$ αντιπροσωπεύει το ποσοστό της ορθότητας της κίνησης. Εάν επιλέχθηκε το σωστό κομμάτι, αυτή η βαθμολογία είναι $0,5d$, όπου d είναι η απόσταση (σε τετράγωνα) μεταξύ του σωστού προορισμού και του επιλεγμένου προορισμού για το κομμάτι. Αν το σωστό τετράγωνο δεχόταν επίθεση αλλά με λάθος κομμάτι, ήταν $0,1$. Στα τελευταία στάδια κάθε διαδρομής (αφού περισσότερα από το 75% των προβλημάτων επιλύθηκαν από τα καλύτερα άτομα), αυτός ο παράγοντας ήταν μόνο $0,0$ ή $1,0$.

- Το N_i είναι το βάθος του προβλήματος. Δεδομένου ότι για μεγαλύτερα N, η εύρεση της κίνησης ζευγαρώματος είναι εκθετικά πιο δύσκολη, αυτός ο παράγοντας αυξάνεται επίσης εκθετικά.

- Boards_i είναι ο αριθμός των σανίδων που εξετάστηκαν από το CRAFTY (μία από τις πρώτες και πιο διάσημες μηχανές σκακιού) για αυτό το πρόβλημα, διαιρεμένος με τον αριθμό που εξετάστηκε από το άτομο. Για τα μικρά N, αυτός ο παράγοντας χρησιμοποιήθηκε μόνο σε μεταγενέστερα στάδια της εξέλιξης.

Ακολουθούν οι τυπικές διαδικασίες αναπαραγωγής, διασταύρωσης και μετάλλαξης και παρατηρείται μία βελτίωση της τάξης του 47% στην ανάπτυξη κόμβων σε σύγκριση με το παγκόσμιας κλάσης πρόγραμμα του CLASY.

Οι Silver et al. [17] γενικεύουν τον αλγόριθμο AlphaGo Zero με σκοπό να μπορεί να αξιοποιηθεί όχι μόνο στο Go αλλά και στο σκάκι και στο shogi (Ιαπωνικό σκάκι). Η υλοποίηση αυτή, όπως και ο AlphaGo Zero, επιτυγχάνει εντυπωσιακά αποτελέσματα αξιοποιώντας τη φιλοσοφία του "Tabula Rasa" και ονομάστηκε AlphaZero. Tabula rasa είναι η λατινική έκφραση του όρου άγραφος πίνακας, έτσι λοιπόν στα πλαίσια αυτής της λογικής ο αλγόριθμος AlphaZero ξεκινώντας από μία τυχαία αρχική κατάσταση και γνωρίζοντας μόνο τους κανόνες του παιχνιδιού και τίποτα άλλο μπορεί αποκλειστικά μέσω της εμπειρίας που αποκτά κάνοντας δοκιμές και σφάλματα να αποκτήσει υπεράνθρωπη ικανότητα παιχνιδιού. Η πλειοψηφία των δυνατότερων προγραμμάτων κατασκευασμένα για να παίζουν σκάκι αξιοποιούν τον αλγόριθμο Apha-Beta ο οποίος είναι ένας αλγόριθμος αναζήτησης που χρησιμοποιείται για να εξερευνήσει το αλγοριθμικό δέντρο του παιχνιδιού και να καθορίσει την καλύτερη κίνηση. Είναι μία βελτιστοποίηση πάνω στον Min-Max αλγόριθμο που "κλαδεύει" τμήματα του αλγοριθμικού δέντρου τα οποία δε χρειάζεται να εξερευνηθούν. Ο AlphaZero όμως αξιοποιεί τη χρήση νευρωνικών δικτύων και μια μέθοδο αναζήτησης δέντρου Monte-Carlo (MCTS) σε αντίθεση με μια εξειδικευμένη αναζήτηση Alpha-Beta. Η αναζήτηση δέντρου Monte Carlo (MCTS) είναι ένας ευρετικός αλγόριθμος αναζήτησης που χρησιμοποιείται συνήθως σε προβλήματα λήψης αποφάσεων με μεγάλο χώρο καταστάσεων. Ο αλγόριθμος χρησιμοποιεί τυχαία δειγματοληψία του δέντρου του παιχνιδιού για να δημιουργήσει μια προσέγγισή του και επιλέγει κινήσεις με βάση τα αποτελέσματα των παιχνιδιών του δείγματος. Με αυτές τις τροποποιήσεις λοιπόν ο AlphaZero μπορεί και αποδίδει καλύτερα από το Stockfish (μια ισχυρή μηχανή σκακιού ανοιχτού κώδικα) στο σκάκι άλλα και από το Elmo (επίσης μια μηχανή σκακιού ανοιχτού κώδικα, που έχει σχεδιαστεί ειδικά για shogi) στο shogi.

Οι Nasa et al. [14] παρουσιάζουν μια υλοποίηση του αλγορίθμου Min-Max αλλά και του Alpha-Beta Pruning για το συνδυαστικό παιχνίδι Connect-4. Το Connect-4 είναι ένα παιχνίδι συνδυαστικής για δύο παίκτες που παίζεται σε έναν κατακόρυφο πίνακα που αποτελείται από επτά στήλες και έξι σειρές. Ο στόχος του παιχνιδιού εί-

ναι ο συνδυασμός τεσσάρων κομματιών, που αντιστοιχούν στον εκάστοτε παίχτη, σε οριζόντια, κάθετη ή διαγώνια γραμμή. Αρχικά χρησιμοποιείται ο αλγόριθμος Min-Max ο οποίος είναι ένας αλγόριθμος λήψης αποφάσεων, υποθέτει ότι και οι δύο παίκτες παίζουν βέλτιστα και στόχος του είναι να βρει την καλύτερη δυνατή κίνηση για τον παίκτη που κάνει την κίνηση. Ο αλγόριθμος Min-Max λειτουργεί δημιουργώντας ένα αλγοριθμικό δέντρο που αντιπροσωπεύει όλες τις πιθανές κινήσεις και τα αποτελέσματά τους. Κάθε κόμβος στο δέντρο αντιπροσωπεύει μια κατάσταση παιχνιδιού και οι ακμές αντιπροσωπεύουν τις πιθανές κινήσεις που μπορούν να γίνουν από αυτήν την κατάσταση. Ο αλγόριθμος αξιολογεί την τιμή κάθε κόμβου χρησιμοποιώντας μία ευρετική διαδικασία που εκτιμά την ποιότητα αυτής της θέσης. Σε κάθε επίπεδο του δέντρου, ο αλγόριθμος εναλλάσσεται μεταξύ της μεγιστοποίησης και της ελαχιστοποίησης των τιμών της διαδικασίας. Όταν είναι η σειρά του παίκτη να μετακινηθεί, ο αλγόριθμος επιλέγει την κίνηση που οδηγεί στην υψηλότερη ευρετική τιμή. Όταν είναι η σειρά του αντιπάλου να κινηθεί, ο αλγόριθμος υποθέτει ότι ο αντίπαλος θα επιλέξει την κίνηση που οδηγεί στη χαμηλότερη ευρετική τιμή. Επαναλαμβάνοντας αυτήν τη διαδικασία αναδρομικά, ο αλγόριθμος μπορεί να καθορίσει την καλύτερη κίνηση που μπορεί να γίνει στην τρέχουσα κατάσταση του παιχνιδιού. Σειρά έχει ο Alpha-Beta Pruning, ο οποίος δεν είναι εντελώς διαφορετικός αλγόριθμος από τον Min-Max αλλά μία βελτιωμένη έκδοσή του. Όπως έχει αναφερθεί και σε άλλο άρθρο παραπάνω είναι μία τεχνική που μειώνει τον αριθμό των κόμβων που πρέπει να αξιολογηθούν στο δέντρο καθώς εξαλείφει τμήματα του δέντρου που δεν μπορούν να οδηγήσουν σε καλύτερη λύση από αυτήν που έχει ήδη βρεθεί. Ο αλγόριθμος λειτουργεί διατηρώντας δύο τιμές, την άλφα και τη βήτα, που αντιπροσωπεύουν τις καλύτερες ευρετικές τιμές των μεγίστων και των ελαχίστων. Αρχικοποιείται το μείον άπειρο στο άλφα και το συν άπειρο στο βήτα. Στη συνέχεια εξερευνάται το δέντρο του παιχνιδιού χρησιμοποιώντας τον αλγόριθμο Min-Max, αλλά με την προσθήκη των τιμών άλφα και βήτα. Σε κάθε επίπεδο του δέντρου, ο αλγόριθμος αξιολογεί τους κόμβους χρησιμοποιώντας τον Min-Max και ενημερώνει τις τιμές άλφα και βήτα καθώς προχωρά. Εάν ο αλγόριθμος βρει έναν κόμβο όπου η καλύτερη μέγιστη τιμή είναι μικρότερη ή ίση με την καλύτερη ελάχιστη τιμή (άλφα geq βήτα), τότε γνωρίζει ότι αυτός ο κλάδος του δέντρου δε θα επιλεγεί ποτέ, επομένως "κλαδεύει" τους κόμβους. Έτσι λοιπόν παρατηρείται

πως οι δύο αλγόριθμοι διαφέρουν τόσο στον απαιτούμενο χρόνο εκτέλεσης όσο και στον αριθμό επαναλήψεων που πραγματοποιούνται με τον Alpha-Beta Pruning να δημιουργεί την κατάσταση του παιχνιδιού πολύ πιο γρήγορα και με πολύ λιγότερες επαναλήψεις.

Οι Clausen et al. [4] αποσκοπούν να βελτιώσουν τον αλγόριθμο AlphaZero, ο οποίος έχει αναφερθεί σε άλλο άρθρο παραπάνω, και να πετύχουν μείωση τόσο στο υπολογιστικό κόστος όσο και στο κόστος υλικού. Αρχικά ο Alpha-Zero τροποποιείται κατάλληλα για να δουλεύει στα μέτρα του παιχνιδιού Connect-4. Δημιουργήθηκαν λοιπόν δύο σετ δοκιμών:

- Το δυνατό, που θεωρεί σωστές μόνο τις κινήσεις που οδηγούν πιο γρήγορα στη νίκη και πιο αργά στην ήττα
- Το αδύναμο, που θεωρεί σωστές τις κινήσεις που φέρουν το ίδιο αποτέλεσμά με την τωρινή κίνηση δίχως όμως να ενδιαφέρεται για το πόσο θα αργήσει να έρθει η νίκη και πόσο γρήγορα μπορεί να έρθει η ήττα.

Στη συνέχεια ομαδοποιήθηκαν οι επαναλαμβανόμενες θέσεις και για να επιταχυνθεί η εκπαίδευση του νευρωνικού δικτύου αξιοποιήθηκε ένας κυκλικός ρυθμός μάθησης καθώς διαφορετικά κομμάτια της εκπαίδευσης ίσως να ευνοηθούν από μία αλλαγή στον ρυθμό εκμάθησης. Επίσης, αξιοποιήθηκε ένα λίγο μεγαλύτερο εύρος εκπαίδευσης και ταυτόχρονα μία μέθοδος πρόβλεψης της επομένης κίνησης του αντιπάλου, αφού παρατηρήθηκε πως σε ορισμένες περιπτώσεις υπήρχαν καλύτερα αποτελέσματα. Τέλος, εφαρμόστηκε η μέθοδος squeeze-and-excitation που επιτρέπει στο δίκτυο να εστιάζει επιλεκτικά στις πιο σχετικές λειτουργίες, ενώ φιλτράρει άσχετες ή "θορυβώδεις" λειτουργίες. Χάρη σε αυτές τις γνωστές βελτιώσεις, ο Alpha-Zero αποδίδει καλύτερα και έχοντας πλέον αυτήν του την εκδοχή ως βάση σύγκρισης οι συγγραφείς προχωράνε στις τρεις κύριες επεκτάσεις που παρουσιάζουν για τον AlphaZero. Αρχικά εφαρμόστηκε η αναζήτηση δέντρου Monte Carlo (MCTS), η οποία έχει προαναφερθεί σε προηγούμενο άρθρο, χάρις την οποία παρατηρήθηκε μια μικρή βελτίωση στα πρώτα στάδια του παιχνιδιού αλλά η βελτίωση που παρέχει αδρανοποιείται από ένα σημείο και μετά. Η δεύτερη προσέγγιση είναι η χρήση εσωτερικών χαρακτηριστικών του δικτύου ως βοηθητικούς στόχους. Το δίκτυο μπορεί να εκπαιδευτεί ώστε να μαθαίνει όχι μόνο την τελική έξοδο του δικτύου αλλά και

ενδιάμεσα χαρακτηριστικά που είναι χρήσιμα για την επίλυση του προβλήματος. Χρησιμοποιώντας τα χαρακτηριστικά αυτά το δίκτυο ενθαρρύνεται να μάθει πιο ενημερωτικές και μεροληπτικές αναπαραστάσεις των δεδομένων εισόδου, οι οποίες μπορούν να βελτιώσουν την απόδοσή του στην εύρεση του κύριου στόχου. Η τελευταία βελτίωση γίνεται στον τομέα του αυτό-παιξίματος (self-play) αξιοποιώντας πορίσματα και αποτελέσματα από προηγούμενες επαναλήψεις για την παραμετροποίηση των επομένων. Εφαρμόζοντας λοιπόν και αυτές τις τρεις επεκτάσεις πάνω στον AlpaZero παρατηρείται μία περαιτέρω βελτίωση, αλλά δυστυχώς μόνο μικρή.

Οι Chisholm και Bradbeer [3] παρουσιάζουν μία βελτίωση με γενετικό αλγόριθμο στη συνάρτηση αξιολόγησης του ταμπλό για ένα πρόγραμμα παιχνιδιού για πούλια (checkers). Η λειτουργία αξιολόγησης του ταμπλό είναι υπεύθυνη για την αξιολόγηση της τρέχουσας κατάστασης του ταμπλό του παιχνιδιού και την ανάθεση ενός αριθμητικού σκορ σε αυτήν τη θέση, που δείχνει πόσο ευνοϊκό είναι για τον παίκτη του οποίου είναι η σειρά. Στη συγκεκριμένη περίπτωση οι συγγραφείς αναθέτουν μία ομάδα τιμών ή ''βαρών'', όπως τα ονομάζουν, για το πολυώνυμο που δίνει το προαναφερόμενο σκορ. Για να δουλέψει αυτό και να δουλέψει η γενετική υλοποίηση που αποσκοπείται, χρησιμοποιείται ως βάση το πρόγραμμα που παίζει συνδυαστικά παιχνίδια, DRAFT5. Το DRAFT5 δημιουργήθηκε για να μπορεί να παίζει ενάντια στον εαυτό του εναλλάσσοντας την αναζήτηση και την κίνηση για τον εκάστοτε ''παίκτη'', όπως η πλειοψηφία των αντίστοιχων προγραμμάτων. Χάρις το συγκεκριμένο χαρακτηριστικό του DRAFT5 μπορεί να συγκρίνει διαφορετικά ''γονίδια'', ή στρατηγικές για εμάς, συγκρίνοντας τις τιμές ή βάρη που τους ανατέθηκαν σαν να στήνει ένα τουρνουά round-robin, δηλαδή ένα τουρνουά που όλοι παίζουν εναντίων όλων. Η συνάρτηση αξιολόγησης του γενετικού αλγορίθμου στη συγκεκριμένη υλοποίηση είναι απλά ο αριθμός των νικών που πέτυχε το κάθε γονίδιο. Χρησιμοποιήθηκε μία απλή και ευέλικτη στρατηγική αναπαραγωγής, μετάλλαξης και διασταύρωσης και εφαρμόστηκε ελιτισμός για να διατηρηθεί το καλύτερο γονίδιο ή στρατηγική από κάθε γενιά. Για περαιτέρω βελτιστοποίηση του αλγορίθμου τα καλύτερα αυτά γονίδια συγκρίνονται μία ακόμη φορά μεταξύ τους στα ίδια πλαίσια ενός τουρνουά round-robin. Τέλος, μία τελική σύγκριση έγινε με τα καλύτερα γονίδια όπως πριν άλλα αυτήν τη φορά ενάντια στην πρωτότυπη έκδοση του DRAFT5. Έτσι συμπεραίνεται πως χωρίς την εισαγωγή οποιονδήποτε δεδομένων

για συγκεκριμένο τομέα, ένας σχετικά απλός γενετικός αλγόριθμος μπορεί να βρει ένα εύλογο σύνολο βαρών αξιολόγησης πίνακα για να παίξει πούλια (checkers).

Οι Oberoi et al. [15] πραγματοποιούν μία υλοποίηση του Extended Classifier System (XCS) πάνω σε ένα ελαφρώς τροποποιημένο ταμπλό (6x6) ξανά για το παιχνίδι συνδυαστικής πούλια (checkers). Ο XCS είναι μια επέκταση του αρχικού αλγορίθμου Classifier Systems (CS), ο οποίος πρωτοεμφανίστηκε από τον John Holland το 1975. Η κύρια διαφορά μεταξύ CS και XCS είναι ότι ο XCS χρησιμοποιεί μια πιο περίπλοκη αναπαράσταση κατηγοριοποιητή και εισάγει πρόσθετα στοιχεία στον αλγόριθμο για να βελτιώσει την απόδοσή του. Στη συγκεκριμένη υλοποίηση αντί για την ίδια την πρόβλεψη, η καταλληλότητα ενός κατηγοριοποιητή καθορίζεται από το πόσο ακριβής είναι. Επίσης, ένας γενετικός αλγόριθμος (GA) και ένα στοιχείο ενισχυτικής μάθησης (RL) χρησιμοποιούνται για διερευνητικούς και μαθησιακούς σκοπούς, αντίστοιχα. Εμβαθύνοντας περισσότερο, στον XCS, υπάρχει μια ομάδα κατηγοριοποιητών, ο καθένας με μια συγκεκριμένη κατάσταση, δράση και αναμενόμενη απόδοση. Όταν το περιβάλλον παρέχει την τρέχουσα κατάσταση συστήματος, οι συνθήκες των κατηγοριοποιητών συγκρίνονται με την τρέχουσα κατάσταση για να σχηματιστεί ένα ''σύνολο αντιστοίχισης''. Εάν ο αριθμός των διαφορετικών ενεργειών στο σετ αγώνα είναι μικρότερος από ένα συγκεκριμένο όριο, δημιουργείται ένας νέος κατηγοριοποιητής με μία πιο γενική συνθήκη. Μετά τον σχηματισμό του σετ αντιστοίχισης, δημιουργείται ένας πίνακας πρόβλεψης για την εκτίμηση της αναμενόμενης απόδοσης για κάθε πιθανή ενέργεια και ο αλγόριθμος επιλέγει την ενέργεια με την υψηλότερη αναμενόμενη απόδοση. Επίσης, ο XCS χρησιμοποιεί έναν συνδυασμό εξερεύνησης και εκμετάλλευσης για να επιλέξει ενέργειες και να ενημερώσει τις τιμές πρόβλεψης των κατηγοριοποιητών με βάση την απόδοση που λαμβάνει από το περιβάλλον. Αυτή η προσέγγιση επιτρέπει στον αλγόριθμο να μαθαίνει και να προσαρμόζεται με την πάροδο του χρόνου για να βελτιώνει την απόδοσή του. Στη βελτίωση του αλγορίθμου συμβάλλουν και οι νέοι κατηγοριοποιητές που δημιουργούνται μέσα από τη χρήση ενός γενετικού αλγορίθμου (GA) που εφαρμόζεται στο σύνολο ενεργειών. Για τον σκοπό αυτό χρησιμοποιούνται γενετικοί τελεστές όπως η μετάλλαξη και η διασταύρωση. Έτσι λοιπόν ο ''πράκτορας'' του XCS κατάφερε να αντεπεξέλθει και να αποδώσει ικανοποιητικά απέναντι σε ανθρώπους τριών διαφορετικών επιπέδων ικανοτήτων αλλά και ενάντια σε ''πράκτορες''

των αλγορίθμων τυχαίας επιλογής και Alpha-Beta.

Ο Thomsen [19] παρουσιάζει τον αλγόριθμο Lambda-Search ή λ-Search, ένας αλγόριθμος που είναι στοχευμένος για χρήση στα συνδυαστικά παιχνίδια σκάκι (chess) και Go με αναζήτηση δέντρων παιχνιδιών δύο τιμών. Ο λ-Search είναι βασισμένος πάνω στην ευρετική διαδικασία null-move pruning (''κλάδεμα'' μηδενικής κίνησης) αλλά επικεντρώνεται στην εύρεση απειλών και την αποτροπή αυτών. Μια κίνηση λ_n είναι μια κίνηση σε ένα παιχνίδι που ικανοποιεί ορισμένες προϋποθέσεις, ανάλογα με το αν είναι η σειρά του επιθετικού ή του αμυντικού. Όταν ο επιτιθέμενος κάνει μια κίνηση λ_n, σημαίνει ότι εάν ο αμυνόμενος δεν μπλοκάρει την κίνηση, υπάρχει τουλάχιστον μία επόμενη κίνηση που θα οδηγήσει στη νίκη του επιτιθέμενου εντός n κινήσεων. Όταν ο αμυνόμενος κάνει μια κίνηση λ_n, σημαίνει ότι δεν υπάρχει καμία μεταγενέστερη κίνηση που θα οδηγήσει στη νίκη του επιτιθέμενου μέσα σε n κινήσεις. Ο λ-Search χρησιμοποιείται για την κατασκευή ενός δέντρου λ_n, ενός δέντρου αναζήτησης που εξετάζει μόνο τις κινήσεις λ_n, για να βοηθήσει στην εύρεση της καλύτερης κίνησης για τον εισβολέα. Το δέντρο λ_n έχει πολύ μικρότερο μέσο παράγοντα διακλάδωσης από το δέντρο πλήρους αναζήτησης, καθιστώντας το πιο αποτελεσματικό. Ο αλγόριθμος λειτουργεί αναδρομικά για να δημιουργήσει μικρότερα λ_n-δέντρα. Έτσι λοιπόν ο Thomsen καταλήγει σε δυο θεωρίες που αποδεικνύουν ότι ο λ-Search δεν είναι μια ευρετική τεχνική κλαδέματος προς τα εμπρός. Αντίθετα όταν η αναζήτηση είναι στη χειρότερη περίπτωση, γίνεται σε ένα ορισμένο βάθος που ονομάζεται λ-$order$ με $n = (d–1)/2$ και επιστρέφει πάντα την τιμή Min-Max (το βέλτιστο αποτέλεσμα για τον παίκτη) εφόσον επιτρέπεται το πάσο και δεν υπάρχει κίνητρο zugzwang (κατάσταση όπου ένας παίκτης αναγκάζεται να κάνει μία μειονεκτική κίνηση). Αυτό σημαίνει ότι ο λ-Search είναι μια αξιόπιστη μέθοδος για την εύρεση της καλύτερης κίνησης σε ένα παιχνίδι, και όχι απλώς μία πρόχειρη εκτίμηση. Μπορεί να μειώσει τον χώρο αναζήτησης σε μεγαλύτερο βαθμό από τον Alpha-Beta και τον Alpha-Beta με null-move pruning και είναι συμβατός με άλλες μεθόδους αναζήτησης, όπως ο Proof-Number Search, επιτρέποντας τον εύκολο συνδυασμό με μηδενικές κινήσεις και αριθμούς απόδειξης.

Οι Yen και Yang [22] παρουσιάζουν μία τροποποίηση του αλγορίθμου αναζήτησης Monte-Carlo (MCTS) και την εφαρμογή του στο παιχνίδι συνδυαστικής Connect-6. Η συγκεκριμένη υλοποίηση αποτελεί μία αναζήτηση Monte-Carlo δύο επιπέδων.

Το πρώτο επίπεδο επικεντρώνεται στο threat space search (TSS), δηλαδή στον έλεγχο των υποψήφιων κινήσεων χρησιμοποιώντας την κατάσταση στην οποία ο αμυνόμενος πρέπει να μπλοκάρει τον επιτιθέμενο, ενώ το δεύτερο επίπεδο επικεντρώνεται στην εύρεση της βέλτιστης κίνησης. Όταν ο επιτιθέμενος εκτελέσει μια κίνηση, ο αμυνόμενος πρέπει να εκτελέσει μία ξεχωριστή αμυντική ενέργεια εάν υπάρχουν περισσότερες από μία απειλές. Για αυτόν το λόγο αξιοποιείται αρχικά ένα double-threat TSS ονομαζόμενο conservative threat space search (CTSS) που πραγματοποιεί συντηρητικές αμυντικές κινήσεις, δηλαδή κινήσεις που περιλαμβάνουν τη διατήρηση μιας ισχυρής αμυντικής θέσης ή τη δημιουργία μιας απειλής στην οποία ο αντίπαλος πρέπει να απαντήσει. Οι κινήσεις αυτές όμως διαφέρουν από τις κανονικές αμυντικές κινήσεις και μακροχρόνια, σε μια παρτίδα, μπορούν να οδηγήσουν σε προβλήματα. Οι συγγραφείς λοιπόν προτείνουν τη χρήση του iterative threat space search (ITSS) που χρησιμοποιεί κανονική άμυνα για κινήσεις διπλής απειλής. Στην πρώτη επανάληψη, ο αλγόριθμος χρησιμοποιεί CTSS για γρήγορη αναζήτηση όλων των πιθανών κινήσεων διπλής απειλής. Εάν το CTSS βρει μια λύση, η αναζήτηση τελειώνει. Διαφορετικά, ο αλγόριθμος προχωρά στη δεύτερη επανάληψη, στην οποία ο αλγόριθμος χρησιμοποιεί κανονική άμυνα για να ανταποκριθεί στις κινήσεις διπλής απειλής που δημιουργούνται στην πρώτη επανάληψη. Αφού κάνει αμυντικές κινήσεις, ο αλγόριθμος δημιουργεί νέες κινήσεις διπλής απειλής με βάση τις απαντήσεις του αντιπάλου. Ταυτόχρονα, το CTSS επαναλαμβάνεται σε αυτές τις νέες κινήσεις διπλής απειλής. Για την εύρεση της βέλτιστης κίνησης αξιοποιείται η τεχνική relevance zone search καθώς μπορεί να χρησιμοποιηθεί όχι μόνο όταν η αμυντική πλευρά αμύνεται ενάντια σε μια λύση CTSS αλλά κι όταν η επιθετική πλευρά αναζητά υποσχόμενες κινήσεις. Όταν η επιθετική πλευρά κάνει μία κίνηση, υπάρχει μία λύση CTSS για τη θέση που προκύπτει. Η τεχνική relevance zone search μπορεί να πραγματοποιηθεί στην επιθετική πλευρά για να προσδιοριστεί εάν η αμυντική πλευρά μπορεί να αμυνθεί ενάντια στη λύση CTSS. Αυτό μπορεί να βοηθήσει την επιθετική πλευρά να εντοπίσει χρήσιμες κινήσεις και να βελτιώσει την αποτελεσματικότητα του αλγορίθμου αναζήτησης. Έτσι ο αλγόριθμος αναζήτησης Monte-Carlo δύο επιπέδων είναι αποτελεσματικότερος από τον απλό αλγόριθμο αναζήτησης Monte-Carlo.

Ο Uiterwijk [20] παρουσιάζει τη λύση του παιχνιδιού Cram με βάση τη θεωρία

των παιχνιδιών συνδυαστικής. Το Cram είναι ένα παιχνίδι στο οποίο δύο παίχτες καλούνται να τοποθετήσουν σε ένα ταμπλό $(m \times n)$ κομμάτια ντόμινο είτε οριζόντια είτε κάθετα. Σκοπός του παιχνιδιού είναι ο ένας από τους δύο παίχτες να φτάσει τον αντίπαλό του σε σημείο που να μην μπορεί να τοποθετήσει άλλο κομμάτι. Η θεωρία των παιχνιδιών συνδυαστικής χωρίζει τα παιχνίδια αυτά σε δυο κατηγορίες, τα partisan games και τα impartial games. Στην πρώτη περίπτωση οι δύο παίκτες έχουν τις ίδιες κινήσεις και μια θέση παιχνιδιού ορίζεται απλώς από το σύνολο όλων των επιλογών της, ενώ στη δεύτερη μπορούμε να αναλύσουμε οποιοδήποτε impartial game ως ένα παιχνίδι Nim. Το αποτέλεσμα του παιχνιδιού καθορίζεται από ένα μαθηματικό αντικείμενο που ονομάζεται ”nimber”, το οποίο είναι ένας μη αρνητικός ακέραιος αριθμός. Το μέγεθος του νούμερου είναι ίσο με την αξία της θέσης στο αρχικό παιχνίδι και έτσι καθορίζονται οι βέλτιστες κινήσεις και στρατηγικές για αυτό το παιχνίδι. Τα πιο γνωστά ”nimbers” στο παιχνίδι Cram είναι τα endgame *0 και το star game *1. Πιο συγκεκριμένα το *1 είναι είναι μια θέση παιχνιδιού όπου είναι δυνατή μόνο μία κίνηση και αυτήν η κίνηση οδηγεί σε μια θέση όπου το παιχνίδι τελειώνει αμέσως με βαθμολογία *0. Ως μέτρο σύγκρισης χρησιμοποιείται μία απλή υλοποίηση ενός Alpha-Beta αλγορίθμου χωρίς καμία ευρετική διαδικασία και γνώση θεωρίας παιχνιδιών συνδυαστικής. Στη συνέχεια αξιοποιείται μία βάση δεδομένων με όλες τις κινήσεις που μπορεί να οδηγήσουν στο τέλος του παιχνιδιού αλλά και δύο ευρετικές διαδικασίες, Fragment Narrowing Heuristic και Fragment Splitting Heuristic, οι οποίες βοηθούν στη διάσπαση του ταμπλό σε μικρότερα, ανεξάρτητα τμήματα, γεγονός που επιτρέπει στον αλγόριθμο αναζήτησης να βρίσκει πιο εύκολα τις βέλτιστες κινήσεις. Τέλος, υλοποιώντας άλλες πέντε μικρές βελτιστοποιήσεις παρατηρείται μία μεγάλη βελτίωση επί του αρχικού Alpha-Beta αλγορίθμου επισημαίνοντας τη σημασία των βάσεων δεδομένων τελικού παιχνιδιού οι οποίες περιέχουν τιμές που υπολογίζονται χρησιμοποιώντας συνδυαστική θεωρία παιγνίων, καθώς αυτές οι βάσεις δεδομένων μπορούν να βοηθήσουν στην καθοδήγηση της διαδικασίας αναζήτησης και να οδηγήσουν σε πιο επιτυχημένες λύσεις. Όμοια υλοποίηση πραγματοποίησαν οι Uiterwijk και Griebel[21], αλλά αυτήν τη φορά στο παιχνίδι Clobber. Δημιουργήθηκε μια βάση δεδομένων που περιέχει τις ακριβείς τιμές CGT για όλα τα υποπαιχνίδια έως και 8 συνδεδεμένες πέτρες. Αυτές οι τιμές συμβάλλουν στη μείωση του χώρου αναζήτησης για τον λύτη, με αποτέλε-

σμα ταχύτερο και πιο αποτελεσματικό παιχνίδι. Ο βαθμός μείωσης εξαρτάται από το μέγεθος του ταμπλό του παιχνιδιού, με μείωση 100% για τους πίνακες στη βάση δεδομένων και 75% για έναν πίνακα 3×6.

Κεφάλαιο 3

Υλοποίηση

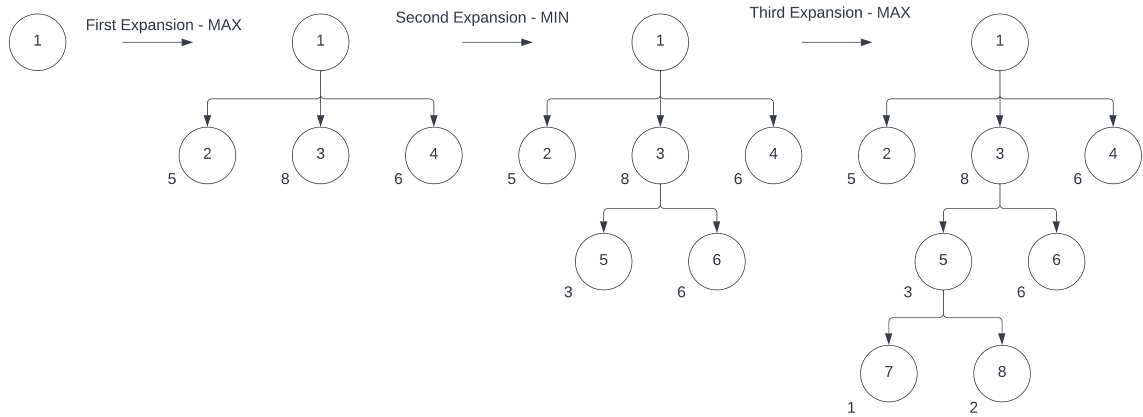
3.1 Τρίλιζα

3.1.1 Min-Max

Για το παιχνίδι της τρίλιζας υλοποιήθηκε ο αλγόριθμος Min-Max του οποίου το αλγοριθμικό δέντρο παρουσιάζεται στο Σχήμα 3.1. Ο κύριος στόχος του αλγορίθμου είναι να καθορίσει την καλύτερη κίνηση ενός παίκτη, δεδομένου ότι ο αντίπαλος του κάνει επίσης τις καλύτερες κινήσεις. Ο αλγόριθμος Min-Max λειτουργεί εναλλάσσοντας τους δύο τύπους αναπαραγωγής ελαχιστοποίησης και μεγιστοποίησης. Υποθέτοντας ότι ο παίκτης μεγιστοποίησης θα αντιδρούσε καλύτερα, ο παίκτης που ελαχιστοποιεί επιλέγει κινήσεις που καταλήγουν σε λιγότερους ''πόντους''. Επίσης, ο παίκτης που ελαχιστοποιεί θα πάρει τις καλύτερες αποφάσεις και ο παίκτης μεγιστοποίησης επιλέγει κινήσεις που οδηγούν σε περισσότερους ''πόντους''. Ο αλγόριθμος ενημερώνει τις βαθμολογίες στην πορεία και διαδίδει τις καλύτερες βαθμολογίες στη ρίζα του δέντρου καθώς ερευνά διάφορες κινήσεις στο δέντρο αναζήτησης. Αυτό επιτρέπει στον αλγόριθμο να αξιολογήσει κάθε πιθανή ενέργεια και απόκριση πριν φτάσει στην κορυφή του δέντρου και κάνει τη βέλτιστη επιλογή.

```
function min_max(node, depth, maximizingPlayer):
    if depth is 0 or node is a terminal node:
        return the heuristic value of node

    if maximizingPlayer:
        bestValue = -INFINITY
        for each child in node's children:
            value = min_max(child, depth - 1, FALSE)
            bestValue = max(bestValue, value)
        return bestValue
    else:  # minimizingPlayer
        bestValue = +INFINITY
        for each child in node's children:
            value = min_max(child, depth - 1, TRUE)
            bestValue = min(bestValue, value)
        return bestValue

# Initial call
bestMove = NULL
bestValue = -INFINITY
for each possible move in currentGameState:
    value = min_max(move, depth, FALSE)
    if value > bestValue:
        bestValue = value
        bestMove = move

return bestMove
```

Αλγόριθμος 1: Min-Max



Σχήμα 3.1: Αλγοριθμικό δέντρο ενός Min-Max αλγόριθμου

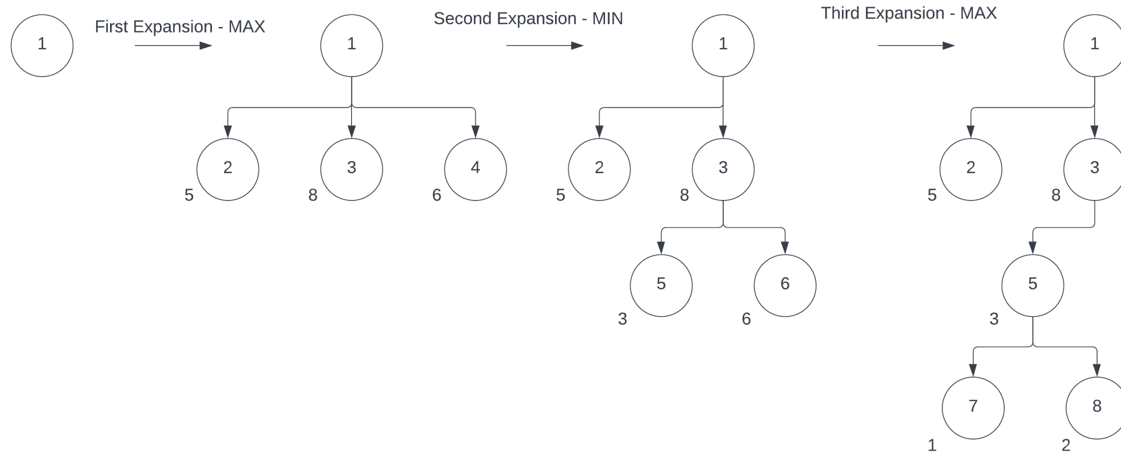
3.2 Connect-4

3.2.1 Aplha-Beta Pruning

Για το παιχνίδι Connect-4 υλοποιήθηκαν δύο αλγόριθμοι, ο Min-Max που προαναφέρθηκε και ο Alpha-Beta Pruning που ουσιαστικά είναι μία επέκταση του Min-Max και η σχηματική του αναπαράσταση δίνεται στο Σχήμα 3.2. Η βασική ιδέα πίσω από τον Alpha-Beta Pruning είναι η μείωση του μεγέθους του δέντρου αναζήτησης ''κλαδεύοντας'' ορισμένα τμήματα του που δεν ικανοποιούν μία συγκεκριμένη συνθήκη. Η συνθήκη αυτή είναι οι τιμές των άλφα και βήτα, εάν η βαθμολογία ενός ''κλαδιού'' δεν ικανοποιεί το εύρος τιμών του άλφα ή του βήτα, μπορεί να κλαδευτεί, μειώνοντας σημαντικά τον αριθμό των κόμβων που πρέπει να αξιολογηθούν.

```
function alphabeta(node, depth, alpha, beta, maximizingPlayer):
    if depth == 0 or node is a terminal node:
        return the heuristic value of node

    if maximizingPlayer:
        value = negative infinity
        for each child in node:
            value = max(value, alphabeta(child, depth - 1,
               alpha, beta, False))
            alpha = max(alpha, value)
            if beta <= alpha:
                break  // Beta cut-off
        return value
    else:
        value = positive infinity
        for each child in node:
            value = min(value, alphabeta(child, depth - 1,
               alpha, beta, True))
            beta = min(beta, value)
            if beta <= alpha:
                break  // Alpha cut-off
        return value
```

Αλγόριθμος 2: Alpha-Beta Pruning



Σχήμα 3.2: Αλγοριθμικό δέντρο ενός Alpha-Beta Pruning αλγόριθμου

3.3 8-Puzzle και Knight Problem

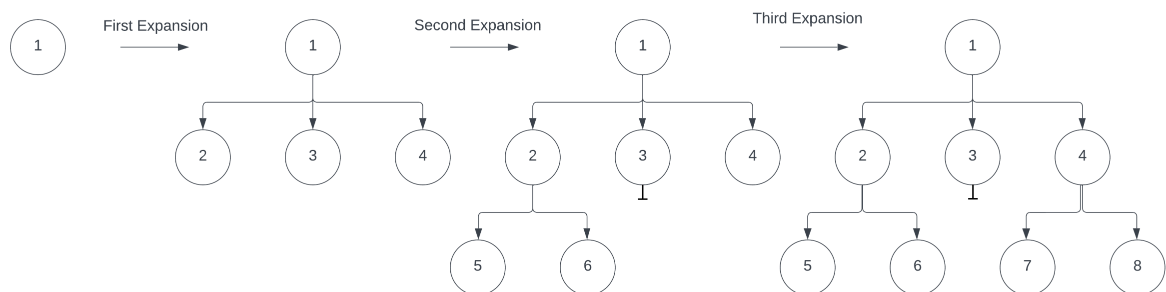
Για τα δύο παιχνίδια 8-Puzzle και Knight Problem υλοποιήθηκαν οι ίδιοι έξι αλγόριθμοι. Αυτό έγινε όχι μόνο για τη μεταξύ τους σύγκριση σε ένα παιχνίδι, αλλά και για να μελετηθεί η συμπεριφορά των ίδιων αλγορίθμων και σε ένα διαφορετικό περιβάλλον, δηλαδή σε ένα άλλο παιχνίδι.

3.3.1 Breadth First Search - BFS

Ο αλγόριθμος αναζήτησης κατά πλάτος ή αλλιώς BFS εξερευνά με επιτυχία το αλγοριθμικό δέντρο επίπεδο προς επίπεδο, όπως παρουσιάζεται στο Σχήμα 3.3. Ξεκινώντας από έναν επιλεγμένο κόμβο πηγής και περνώντας από όλους τους κόμβους παιδιά, ελέγχει από τα αριστερά προς τα δεξιά κάθε κόμβο στο επίπεδο και στη συνέχεια προχωράει στο επόμενο. Υλοποιείται χρησιμοποιώντας μια δομή δεδομένων ουράς, καθιστώντας τον αποδοτικό ως προς την πολυπλοκότητα του χρόνου για μία πληθώρα σχετικά απλών προβλημάτων.

```
function BFS(graph, startNode, targetNode):
    initialize an empty queue
    create a set to keep track of visited nodes

    enqueue the startNode into the queue
    mark the startNode as visited and add it to the visited set

    while the queue is not empty:
        current = dequeue from the queue
        if current is the targetNode:
            return "Target found"

        for each neighbor of current:
            if neighbor is not in the visited set:
                enqueue neighbor into the queue
                mark neighbor as visited and add it
                to the visited set

    return "Target not found"
```

Αλγόριθμος 3: Breadth First Search - BFS



Σχήμα 3.3: Αλγοριθμικό δέντρο ενός Breadth First Search αλγόριθμου

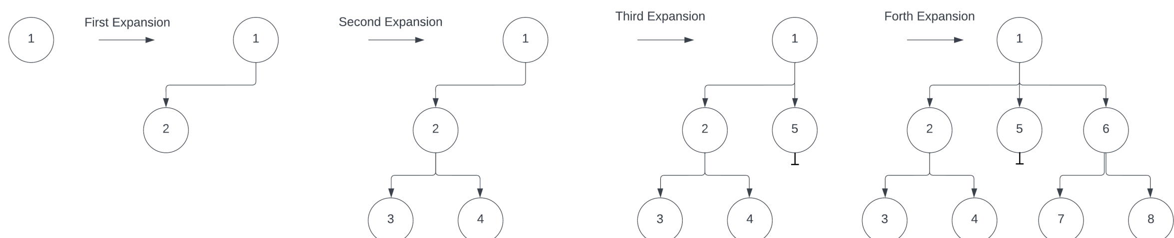
3.3.2 Depth First Search - DFS

Ο αλγόριθμος αναζήτησης κατά βάθος ή αλλιώς DFS αυτήν τη φορά εξερευνά το αλγοριθμικό δέντρο ”κλαδί” προς ”κλαδί”, όπως παρουσιάζεται στο Σχήμα 3.4. Ομοίως ξεκινώντας από έναν επιλεγμένο κόμβο πηγής περνάει αυτήν τη φορά από

όλα τα επίπεδα του πρώτου κόμβου παιδί, ελέγχοντας δηλαδή τους πιο αριστερούς κόμβους κάθε φορά, και στη συνέχεια επιστρέφει και κάνει το ίδιο για το αμέσως δεξιά κόμβο παιδί. Ο συγκεκριμένος υλοποιείται με μία δομή δεδομένων στοίβας ή με αναδρομή και οι επιδόσεις του μπορεί να είναι κοντά σε αυτές του BFS ή ακόμα και να αποκλίνουν κατά πολύ, ανάλογα με το πρόβλημα.

```
function DFS(graph, startNode, targetNode):
    create an empty stack
    create a set to keep track of visited nodes

    push startNode onto the stack
    mark startNode as visited and add it to the visited set

    while the stack is not empty:
        current = pop from the stack
        if current is targetNode:
            return "Target found"

        for each neighbor of current in graph:
            if neighbor is not in visited:
                push neighbor onto the stack
                mark neighbor as visited and add it
                to the visited set

    return "Target not found"
```

Αλγόριθμος 4: Depth First Search - DFS



Σχήμα 3.4: Αλγοριθμικό δέντρο ενός Depth First Search αλγόριθμου

3.3.3 A* Search (Manhattan distance)

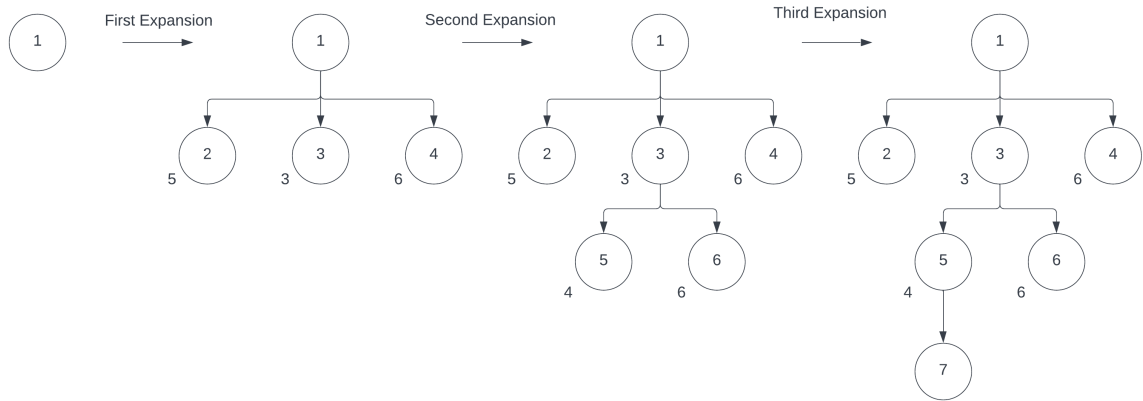
Ο A* και πιο συγκεκριμένα ο A* με την ευρετική διαδικασία απόστασης Manhattan είναι ένας από τους πιο γνωστούς αλγορίθμους εύρεσης συντομότερου μονοπατιού, η ανάπτυξη του αλγοριθμικού του δέντρου παρουσιάζεται στο Σχήμα 3.5. Η ευρετική διαδικασία απόστασης Manhattan υπολογίζει το άθροισμα των απόλυτων τιμών των διαφορών των οριζοντίων και κάθετων συντεταγμένων του εκάστοτε κόμβου σε σχέση με τον κόμβο στόχο. Με αυτόν τον τρόπο, ο αλγόριθμος επιλέγει να αναπτύξει τον κόμβο παιδί με το μικρότερο άθροισμα απόστασης Manhattan και κόστος βήματος. Καθίσταται λοιπόν ως ένας πολύ αποτελεσματικός αλγόριθμος, καθώς αποφεύγει την ανάπτυξη περιττών κόμβων.

```
1  function calculateManhattanDistance(node1, node2):
2      return abs(node1.x - node2.x) + abs(node1.y - node2.y)
3
4  function reconstructPath(node):
5      path = []
6      while node is not null:
7          prepend node to path
8          node = parent of node
9      return path
```

Αλγόριθμος 5: A* Search (Manhattan distance) - Part 1

```
1   function AStar(graph, startNode, targetNode):
2       create open set (priority queue) for nodes to be evaluated
3       create closed set (set of visited nodes)
4       initialize startNode's g-score to 0
5       initialize startNode's f-score using Manhattan distance
           heuristic
6       enqueue startNode with f-score into open set
7   
8       while open set is not empty:
9           current = dequeue from open set (node with lowest
               f-score)
10          if current is targetNode:
11              return reconstructPath(current)
12          add current to closed set
13          for each neighbor of current:
14              if neighbor is in closed set:
15                  continue  # skip already evaluated neighbors
16              tentative_g_score = g-score of current + cost
                   between current and neighbor
17              if neighbor is not in open set or tentative_g_score
                   < g-score of neighbor:
18                  set g-score of neighbor to tentative_g_score
19                  calculate Manhattan distance heuristic from
                       neighbor to targetNode
20                  set f-score of neighbor to g-score of neighbor +
                       Manhattan distance heuristic
21                  set parent of neighbor to current
22                  if neighbor is not in open set:
23                      enqueue neighbor with f-score into open set
24      return "No path found"
```

Αλγόριθμος 6: A* Search (Manhattan distance) - Part 2



Σχήμα 3.5: Αλγοριθμικό δέντρο ενός A* Search (Manhattan distance) αλγόριθμου

3.3.4 A* Search (Manhattan distance and reversal penalty)

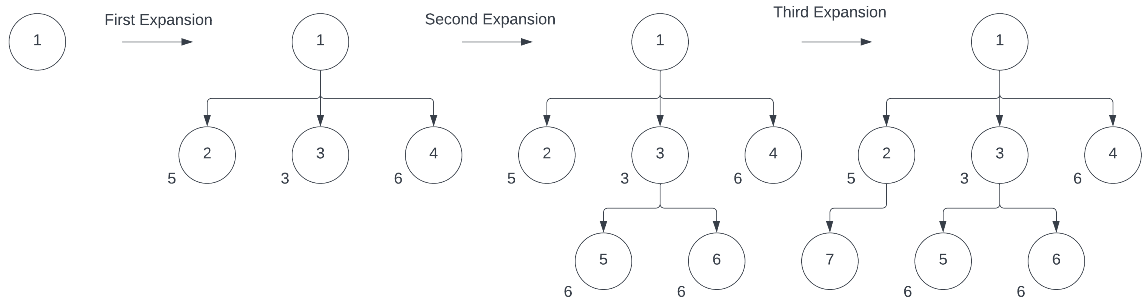
Η συγκεκριμένη εκδοχή του A* είναι όμοια με την προαναφερομένη αλλά απαρτίζεται από έναν επιπρόσθετο παράγοντα, μία ποινή κόστους για τους κόμβους που θα αλλάξουν την κατεύθυνση κίνησης του πράκτορα, η ανάπτυξη του αλγοριθμικού του δέντρου παρουσιάζεται στο Σχήμα 3.6. Αυτό έχει ως αποτέλεσμα την αποφυγή ακόμα περισσότερων κόμβων που σε πολλές περιπτώσεις μπορεί να αποτελέσει σημαντικό παράγοντα στη βελτίωση της απόδοσης του αλγορίθμου.

```
function calculateManhattanDistance(node1, node2):
    return abs(node1.x - node2.x) + abs(node1.y - node2.y)

function applyReversalPenalty(neighbor, current, target):
    reversalPenalty = 0  # Adjust this penalty value as needed
    if parent of current is not null:
        parentDirection = direction from parent of current to
           current
        neighborDirection = direction from current to neighbor
        if parentDirection is opposite of neighborDirection:
            reversalPenalty = 1  # Or any other appropriate
               penalty
    return reversalPenalty

function reconstructPath(node):
path = []
while node is not null:
    prepend node to path
    node = parent of node
return path
```

Αλγόριθμος 7: A* Search (Manhattan distance and reversal penalty) - Part 1

```
1   function AStar(graph, startNode, targetNode):
2       create an open set (priority queue) for nodes to be evaluated
3       create a closed set (set of visited nodes)
4       initialize the startNode's g-score to 0 (the cost from start
           to start is 0)
5       initialize the startNode's f-score using Manhattan distance
           heuristic
6       enqueue startNode with f-score into the open set
7
8       while the open set is not empty:
9           current = dequeue from the open set (node with lowest
               f-score)
10          if current is targetNode:
11              return reconstructPath(current)
12          add current to the closed set
13          for each neighbor of current:
14              if neighbor is in the closed set:
15                  continue  # skip already evaluated neighbors
16              tentative_g_score = g-score of current + cost
                   between current and neighbor
17              if neighbor is not in the open set or
                   tentative_g_score < g-score of neighbor:
18                  set g-score of neighbor to tentative_g_score
19                  calculate Manhattan distance heuristic from
                       neighbor to targetNode
20                  apply reversal penalty if applicable
21                  set f-score of neighbor to g-score of neighbor +
                       Manhattan distance heuristic
22                  set parent of neighbor to current
23                  if neighbor is not in the open set:
24                      enqueue neighbor with f-score into the open
                           set
25      return "No path found"
```

Αλγόριθμος 8: A* Search (Manhattan distance and reversal penalty) - Part 2



Σχήμα 3.6: Αλγοριθμικό δέντρο ενός A* Search (Manhattan distance and reversal penalty) αλγόριθμου

3.3.5 Branch and Bound - B&B

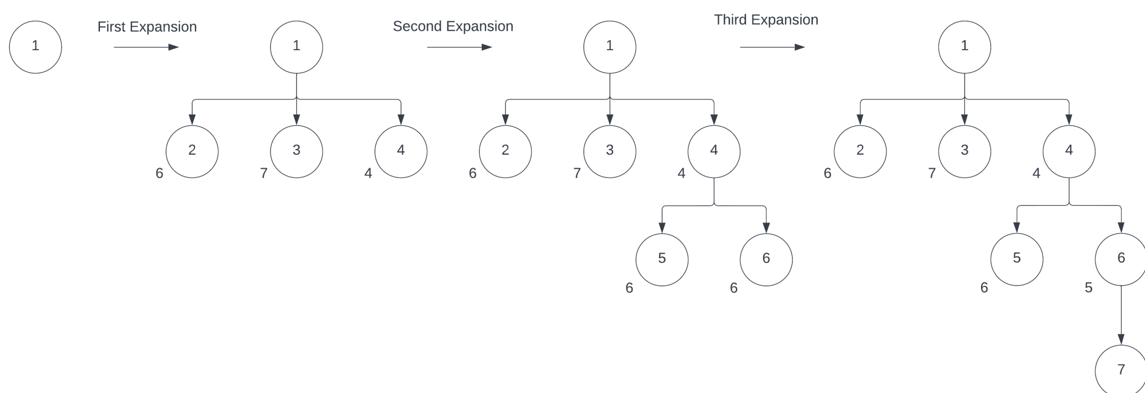
Ο αλγόριθμος Branch and Bound ή αλλιώς B&B χρησιμοποιείται για τη μεθοδική αναζήτηση της βέλτιστης λύσης σε ένα αλγοριθμικό δέντρο, το οποίο και αναπτύσσει όπως παρουσιάζει το Σχήμα 3.7. Χωρίζει το δέντρο σε μικρότερα κομμάτια "κλαδιά", υπολογίζει όρια για να μετρήσει την ποιότητα των υποψήφιων λύσεων εκ των οποίων οι υποδεέστερες "κλαδεύονται", αποφεύγοντας έτσι την εξερεύνηση περιττών μονοπατιών. Με τη σταδιακή μείωση του χώρου αναζήτησης, είναι εφικτή η ταχεία ανακάλυψη βέλτιστων λύσεων.

```
function calculateBound(node):
    // Calculate a bound for the given node based on the
       problem's specifics
    // For minimization problems, it should be an upper bound
       (node.bound >= node.solution)
    // For maximization problems, it should be a lower bound
       (node.bound <= node.solution)
    // Use problem-specific heuristics or relaxations to
       estimate the bound

function prune(node):
    // Skip this node and its subtree if it can be pruned based
       on bounds

function createChildNodes(node):
    // Generate child nodes by branching from the current node
    // Based on problem-specific branching rules

function isLeafNode(node):
    // Determine if the node is a leaf node (no further
       branching possible)
```

Αλγόριθμος 9: Branch and Bound - B&B - Part 1

```
1  function BranchAndBound(problem):
2      create an empty priority queue (min-heap) Q
3      initialize the bestSolution to a large initial value
          (positive infinity for minimization problems, negative
          infinity for maximization problems)
4      create a root node representing the initial state
5      calculate an initial bound for the root node and set it as
          the root's bound
6      enqueue the root node into Q
7      while Q is not empty:
8          node = dequeue the highest priority node from Q
9          if node.bound is worse than the current bestSolution:
10             prune this node (skip it)
11         else if node is a leaf node:
12             update bestSolution if the node's solution is better
13         else:
14             create child nodes by branching from the current node
15             calculate bounds for each child node
16             enqueue the child nodes into Q
17     return bestSolution
```

Αλγόριθμος 10: Branch and Bound - B&B - Part 2



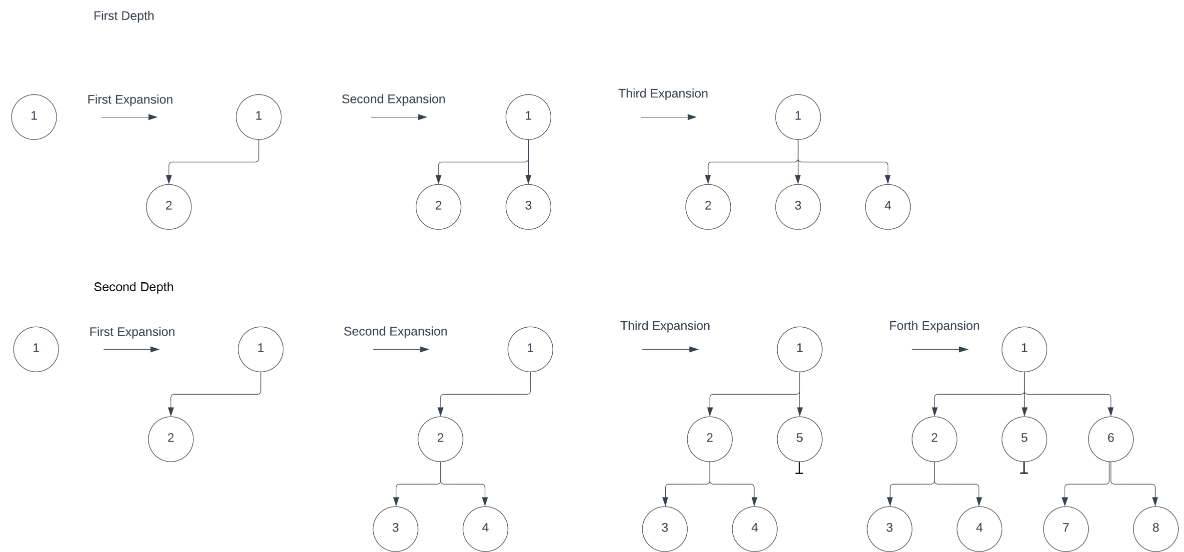
Σχήμα 3.7: Αλγοριθμικό δέντρο ενός Branch and Bound αλγόριθμου

3.3.6 Iterative Deepening Search - IDS

Ο αλγόριθμος αναζήτησης επαναληπτικής εμβάθυνσης ή αλλιώς IDS χρησιμοποιείται, όπως και οι υπόλοιποι αλγόριθμοι που προαναφέρθηκαν, για τον αποτελεσματικό εντοπισμό της καλύτερης λύσης σε ένα δέντρο αναζήτησης, το οποίο και αναπτύσσει όπως παρουσιάζει το Σχήμα 3.8. Ο αλγόριθμος ξεκινάει με τον ορισμό ενός μέτριο ορίου βάθους το οποίο σταδιακά αυξάνεται μέχρι να βρεθεί η βέλτιστη λύση. Αυτό διασφαλίζει ότι αναζητά βαθύτερα επίπεδα, μόνο εάν είναι απολύτως απαραίτητο για τον εντοπισμό της καλύτερης λύσης, καθιστώντας τον ιδιαίτερα χρήσιμο για καταστάσεις αναζήτησης που το βάθος της λύσης είναι αβέβαιο.

```
function IterativeDeepeningSearch(root, target):
    depth_limit = 0

    while true:
        result = DepthLimitedSearch(root, target, depth_limit)
        if result is "found":
            return "Target found"
        else if result is "not found":
            return "Target not found"

        depth_limit = depth_limit + 1

function DepthLimitedSearch(node, target, depth_limit):
    if node is target:
        return "found"

    if depth_limit is 0:
        return "not found"

    if node is a leaf node:
        return "not found"

    for each child of node:
        result = DepthLimitedSearch(child, target, depth_limit -
           1)
        if result is "found":
            return "found"

    return "not found"
```

Αλγόριθμος 11: Iterative Deepening Search - IDS



Σχήμα 3.8: Αλγοριθμικό δέντρο ενός Iterative Deepening Search αλγόριθμου

Κεφάλαιο 4

Υπολογιστική μελέτη

Η υλοποίησή του κώδικα έγινε χρησιμοποιώντας τη γλώσσα προγραμματισμού Python και το σύνολο του αποτελείται από εννέα αρχεία. Ένα από αυτά τα αρχεία με όνομα `Games.py` περιέχει τη main συνάρτηση, τα δύο αρχεία `Tic_Tac_Toe_Players.py` και `Tic_Tac_Toe_Algorithm.py` αποτελούν την υλοποίηση της τρίλιζας, τα δυο αρχεία `Connect_4_Players.py` και `Connect_4_Board.py` αποτελούν την υλοποίηση του Connect-4, τα δύο αρχεία `Eight_Puzzle_Node.py` και `Eight_Puzzle_Algorithms.py` αποτελούν την υλοποίηση του 8-Puzzle και τα δύο αρχεία `The_Knight_Problem_Node.py` και `The_Knight_Problem_Algorithms.py` αποτελούν την υλοποίηση του Knight Problem.

4.1 Μετρικές και Προβλήματα

Τα προβλήματα τα οποία μελετούνται είναι τέσσερα, η απόδοση ενός πράκτορα στο παιχνίδι της τρίλιζας, η σύγκριση δύο αλγορίθμων στο παιχνίδι Connect-4, η επίλυση του 8-Puzzle και η εύρεση του συντομότερου μονοπατιού για το πιόνι του ιππότη πάνω στο ταμπλό μιας σκακιέρας ($N \times N$). Για την πρώτη περίπτωση, αυτή της τρίλιζας, ο πράκτορας έρχεται αντιμέτωπος με τον χρήστη και με βάση το αν νικάει, έρχεται ισοπαλία ή χάνει κρίνεται η επίδοσή του. Για τη δεύτερη περίπτωση, οι δύο αλγόριθμοι έρχονται αντιμέτωποι τόσο με τον εαυτό τους αλλά και μεταξύ τους ώστε να μπορούν να γίνουν συγκρίσεις τόσο στο ποιος νικάει με βάση το ορισμένο βάθος αναζήτησης άλλα και σε πόσο χρόνο. Για την τρίτη περίπτωση, του 8-Puzzle, δημιουργούνται εκατό διαφορετικά τέτοιου τύπου προβλήματα με τυχαία αρχική κατάσταση και ίδια κατάσταση στόχου έτσι ώστε να γίνουν συγκρίσεις τόσο στον χρόνο εκτέλεσης όσο και στον αριθμό των κόμβων που επεκτάθηκαν από τους αλγορίθμους που θα επιχειρήσουν να τα λύσουν. Ομοίως, για την τέταρτη περί-

πτωση, του Knight Problem, ακολουθείται η ίδια μεθοδολογία για να γίνει σύγκριση των μετρήσεων χρόνων εκτέλεσης και αριθμών επεκτεινόμενων κόμβων.

Αξίζει να αναφερθεί ότι οι δοκιμές και οι μετρήσεις έγιναν αξιοποιώντας το Visual Studio Code σε σύστημα με τα εξής χαρακτηριστικά:

- Επεξεργαστής: AMD Ryzen 7 1700 (overclocked στα 3.5GHz), επεξεργαστής 8 πυρήνων για Socket AM4.
- Μνήμη Ram: G.Skill Trident Z 16GB DDR4 με 2 Modules (2x8GB) και ταχύτητα 3200Hz.

4.2 Αποτελέσματα πειραμάτων

4.2.1 Τρίλιζα

Το πόρισμα που προκύπτει μετά από μία πληθώρα δοκίμων της τρίλιζας, είναι πως ο Min-Max αλγόριθμος που χρησιμοποιείται για την υλοποίησή της είναι αήττητος. Δηλαδή, στη χειρότερη περίπτωση θα φέρει ισοπαλία, ανάλογα με τις ικανότητες του αντιπάλου του. Για τον λόγο αυτό είναι κρίθηκε περιττή η παρουσίαση των αποτελεσμάτων για αυτό το παιχνίδι.

4.2.2 Connect-4

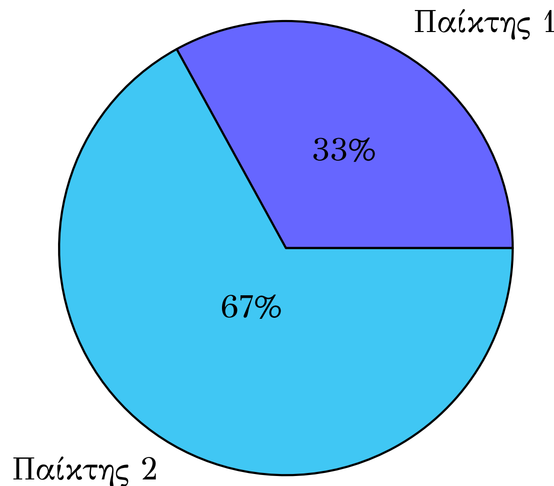
Στο Connect-4 υπάρχουν τέσσερις περιπτώσεις, Alpha-Beta ενάντια Alpha-Beta, Min-Max ενάντια Min-Max, Min-Max ενάντια Alpha-Beta και Alpha-Beta ενάντια Min-Max. Στις τέσσερις αυτές περιπτώσεις γίνεται αύξηση στο βάθος αναζήτησης στο οποίο έχουν πρόσβαση οι αλγόριθμοι ξεκινώντας από το 1 μέχρι και το 6 και παρατηρείται ποιος πράκτορας νικάει κάθε φορά και ποιος ήταν ο χρόνος εκτέλεσης, δηλαδή πόσο διήρκησε το παιχνίδι.

Ο Πίνακας 4.1 παρουσιάζει ποιος παίκτης νίκησε σε κάθε ένα από τα διαφορετικά βάθη αναζήτησης για την περίπτωση Alpha-Beta ενάντια Alpha-Beta και ταυτόχρονα τον χρόνο εκτέλεσης για κάθε βάθος αναζήτησης, αντίστοιχα.

Στη συνέχεια στο Σχήμα 4.1 παρουσιάζεται η αναλογία νικών μεταξύ των δυο παικτών για την περίπτωση Alpha-Beta ενάντια Alpha-Beta.

Πίνακας 4.1: Alpha-Beta ενάντια Alpha-Beta

Βάθος	**Νικητής**	**Χρόνος Εκτέλεσης**
1	Παίκτης 2	0.043
2	Παίκτης 2	0.302
3	Παίκτης 2	0.258
4	Παίκτης 1	1.435
5	Παίκτης 1	3.649
6	Παίκτης 2	19.610



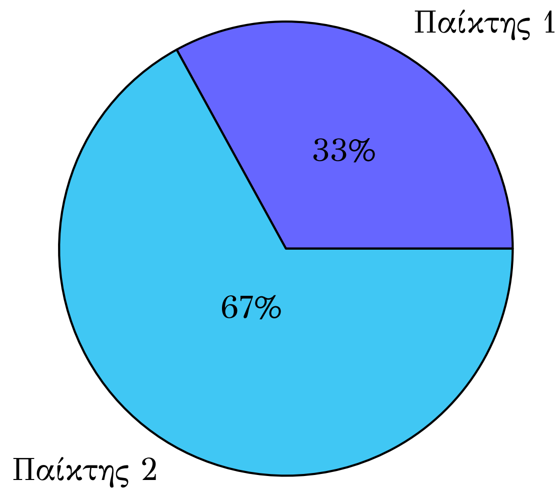
Σχήμα 4.1: Αναλογία νικών μεταξύ παικτών για Alpha-Beta ενάντια Alpha-Beta

Ο Πίνακας 4.2 παρουσιάζει ποιος παίκτης νίκησε σε κάθε ένα από τα διαφορετικά βάθη αναζήτησης για την περίπτωση Min-Max ενάντια Min-Max και ταυτόχρονα τον χρόνο εκτέλεσης για κάθε βάθος αναζήτησης, αντίστοιχα.

Πίνακας 4.2: Min-Max ενάντια Min-Max

Βάθος	**Νικητής**	**Χρόνος Εκτέλεσης**
1	Παίκτης 2	0.046
2	Παίκτης 2	0.544
3	Παίκτης 2	1.120
4	Παίκτης 1	15.328
5	Παίκτης 1	47.680
6	Παίκτης 2	732.541

Στη συνέχεια στο Σχήμα 4.2 παρουσιάζεται η αναλογία νικών μεταξύ των δυο παικτών για την περίπτωση Min-Max ενάντια Min-Max.



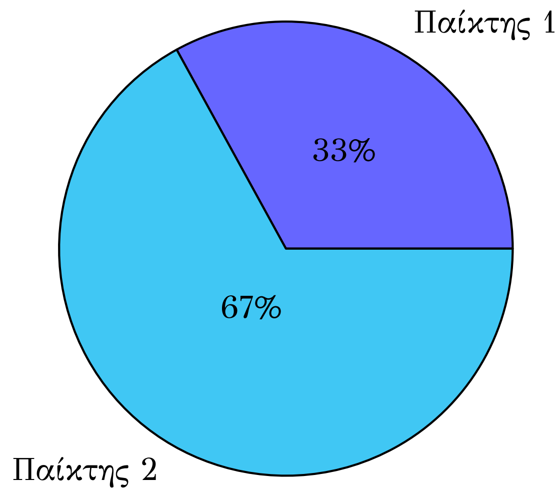
Σχήμα 4.2: Αναλογία νικών μεταξύ παικτών για Min-Max ενάντια Min-Max

Ο Πίνακας 4.3 παρουσιάζει ποιος παίκτης νίκησε σε κάθε ένα από τα διαφορετικά βάθη αναζήτησης για την περίπτωση Min-Max ενάντια Alpha-Beta και ταυτόχρονα τον χρόνο εκτέλεσης για κάθε βάθος αναζήτησης, αντίστοιχα.

Πίνακας 4.3: Min-Max ενάντια Alpha-Beta

Βάθος	**Νικητής**	**Χρόνος Εκτέλεσης**
1	Παίκτης 2	0.045
2	Παίκτης 2	0.453
3	Παίκτης 2	0.716
4	Παίκτης 1	8.883
5	Παίκτης 1	28.244
6	Παίκτης 2	390.998

Στη συνέχεια στο Σχήμα 4.3 παρουσιάζεται η αναλογία νικών μεταξύ των δυο παικτών για την περίπτωση Min-Max ενάντια Alpha-Beta.



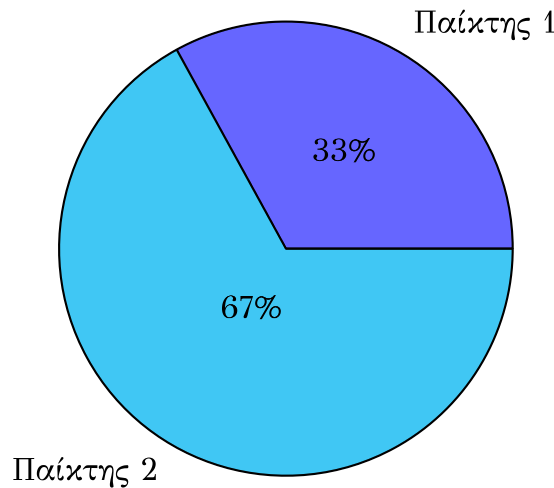
Σχήμα 4.3: Αναλογία νικών μεταξύ παικτών για Min-Max ενάντια Alpha-Beta

Ο Πίνακας 4.4 παρουσιάζει ποιος παίκτης νίκησε σε κάθε ένα από τα διαφορετικά βάθη αναζήτησης για την περίπτωση Alpha-Beta ενάντια Min-Max και ταυτόχρονα τον χρόνο εκτέλεσης για κάθε βάθος αναζήτησης, αντίστοιχα.

Πίνακας 4.4: Alpha-Beta ενάντια Min-Max

Βάθος	**Νικητής**	**Χρόνος Εκτέλεσης**
1	Παίκτης 2	0.044
2	Παίκτης 2	0.422
3	Παίκτης 2	0.687
4	Παίκτης 1	8.108
5	Παίκτης 1	22.940
6	Παίκτης 2	366.547

Στη συνέχεια στο Σχήμα 4.4 παρουσιάζεται η αναλογία νικών μεταξύ των δυο παικτών για την περίπτωση Alpha-Beta ενάντια Min-Max.



Σχήμα 4.4: Αναλογία νικών μεταξύ παικτών για Alpha-Beta ενάντια Min-Max

Εύκολα μπορεί να προσέξει κανείς πως οι νικητές ανά βάθος είναι οι ίδιοι και στις τέσσερις περιπτώσεις, με αποτέλεσμα και η αναλογία νικών να είναι ίδια. Αυτό είναι αναμενόμενο όμως, καθώς ο αλγόριθμος Alpha-Beta Pruning είναι μία επέκταση του Min-Max που σημαίνει ότι τα αποτελέσματα του θα είναι ίδια με αυτά του Min-Max αλλά ταχύτερα, όπως και αποδεικνύεται από τους χρόνους εκτέλεσης.

Αξίζει να αναφερθεί ότι για την περίπτωση Alpha-Beta ενάντια Alpha-Beta έγιναν και τρεις ακόμα δοκιμές καθώς ήταν η λιγότερο χρονοβόρα περίπτωση και για βάθος επτά νίκησε ο παίκτης δύο σε 89.031 δευτερόλεπτα, για βάθος οκτώ νίκησε ο παίκτης ένα σε 35.878 δευτερόλεπτα και τέλος για βάθος εννέα νίκησε ο παίκτης 2 σε 1,933.473 δευτερόλεπτα

4.2.3 8-Puzzle

Στο 8-Puzzle, με σκοπό τη σύγκριση της συμπεριφοράς μεταξύ των αλγορίθμων BFS, DFS, A* (με Manhattan Distance ως Heuristic), A* (με Manhattan Distance ως Heuristic αλλά και Reversal Penalty), B&B και IDS, έγινε ένας μεγάλος αριθμός μετρήσεων και καταγράφηκαν εκατό εκτελέσεις. Για κάθε μία από τις εκατό εκτελέσεις παρουσιάζονται οι χρόνοι εκτέλεσης των έξι αλγορίθμων καθώς και ο αριθμός των κόμβων που χρειάστηκε να επεκταθούν μέχρι να φτάσουν το 8-Puzzle στην κατάσταση στόχο. Αξίζει να αναφερθεί πως κάθε μία από τις εκατό εκτελέσεις ορίζεται από μία συγκεκριμένη αρχική κατάσταση του 8-Puzzle, από την οποία οι αλγόριθμοι φτάνουν στην κατάσταση στόχο και μόνο μόλις τρέξουν και οι έξι αλγό-

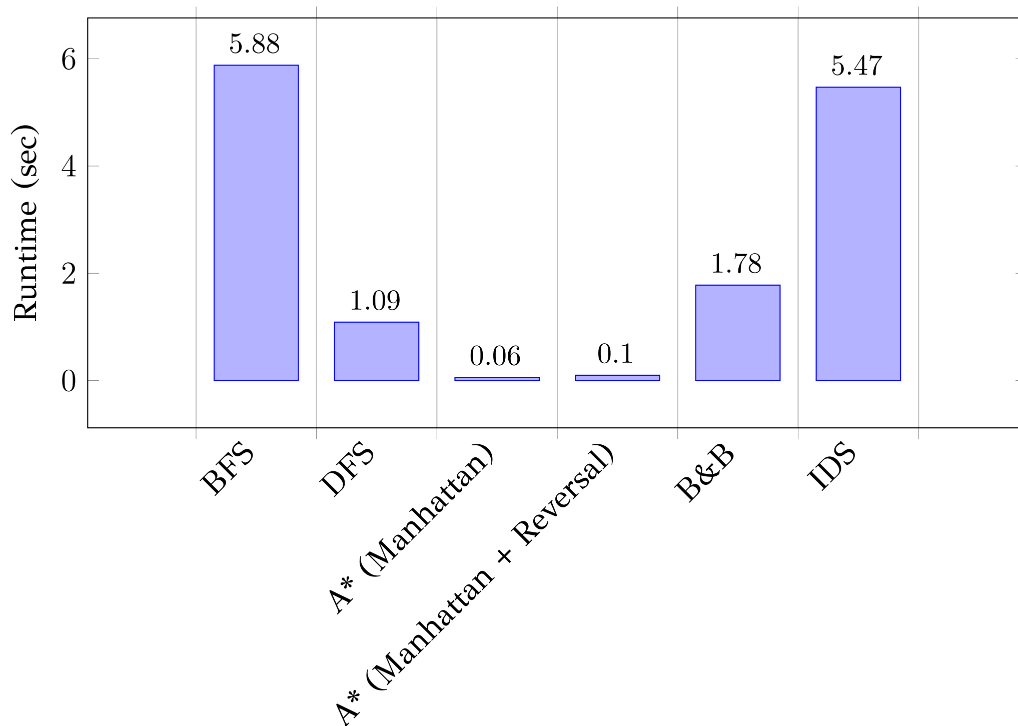
ριθμοι αλλάζει αυτή η αρχική κατάσταση, ώστε να διασφαλιστεί πως οι μετρήσεις των αλγορίθμων σε κάθε εκτέλεση γίνονται με βάση τα ίδια δεδομένα.

Ο Πίνακας 4.5 παρουσιάζει τον μέσο χρόνο εκτέλεσης αλλά και τον μέσο αριθμό κόμβων που επεκτάθηκαν, για κάθε έναν από τους έξι αλγορίθμους στις εκατό καταγεγραμμένες εκτελέσεις του 8-Puzzle.

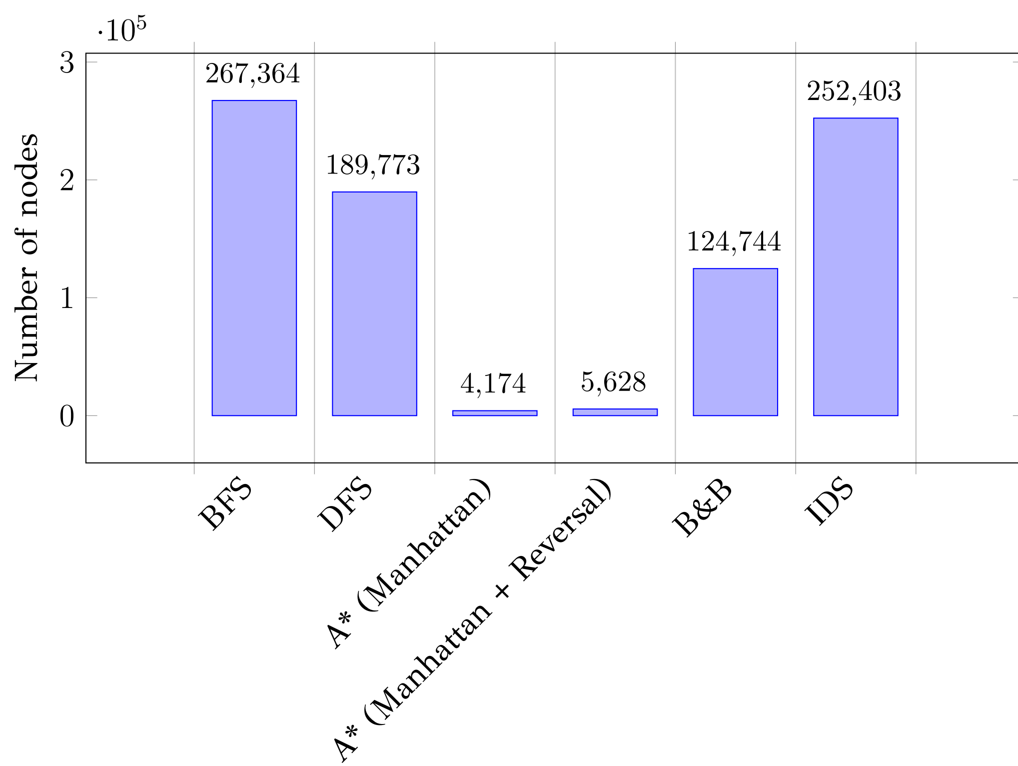
Πίνακας 4.5: Μέσοι όροι του 8-Puzzle.

Αλγόριθμος	**Χρόνος Εκτέλεσης**	**Αριθμός Κόμβων**
BFS	5.88	267,364
DFS	1.09	189,773
A* (Manhattan)	0.06	4,174
A* (Manhattan + Reversal)	0.10	5,628
B&B	1.78	124,744
IDS	5.47	252,403

Στη συνέχεια, τα Σχήματα 4.5 και 4.6 απεικονίζουν τους μέσους όρους των χρόνων εκτέλεσης και αριθμών κόμβων, αντίστοιχα, που παρουσιάζονται στον Πίνακα 4.5.



Σχήμα 4.5: Μέσος όρος χρόνου εκτέλεσης από 100 μετρήσεις



Σχήμα 4.6: Μέσος όρος αριθμού κόμβων που επεκτάθηκαν από 100 μετρήσεις

Παρατηρώντας τον Πίνακα 4.5 αλλά και τα Σχήματα 4.5 και 4.6 είναι εμφανές πως για το συγκεκριμένο πρόβλημα, από τους αλγορίθμους που δε χρησιμοποιούν κάποια ευρετική διαδικασία, ο DFS ευνοείται σε σύγκριση με τους BFS και IDS καθώς έχει τον μικρότερο χρόνο εκτέλεσης μεταξύ των τριών αλλά και το μικρότερο αριθμό κόμβων που επεκτάθηκαν. Από τους αλγορίθμους που χρησιμοποιούν κάποια ευρετική διαδικασία, ο B&B όχι μόνο είναι ο πιο αργός αλλά επεκτείνει και τον μεγαλύτερο αριθμό κόμβων. Αυτό, όμως, είναι λογικό καθώς η ευρετική του διαδικασία είναι η λιγότερο ενημερωμένη. Τέλος, οι δύο A* είναι οι ιδανικότεροι αλγόριθμοι σε σύγκριση με τους άλλους τέσσερις που χρησιμοποιήθηκαν καθώς έχουν τον μικρότερο χρόνο εκτέλεσης αλλά επεκτείνουν και τον μικρότερο αριθμό κόμβων. Αξίζει να σημειωθεί ότι παρά το γεγονός ότι ο A* (Manhattan Distance plus Reversal Penalty) έχει πιο ενημερωμένη ευρετική διαδικασία αποδίδει λίγο χειρότερα από τον A* (Manhattan Distance) καθώς στο συγκεκριμένο πρόβλημα πολλές φορές είναι καλύτερο να αλλάξεις κατεύθυνση πάρα να συνεχίσεις στην ίδια.

4.2.4 Knight Problem

Το Knight Problem, κυμαίνεται περίπου στα ίδια πλαίσια με το 8-Puzzle. Δηλαδή, γίνεται μία σύγκριση συμπεριφοράς μεταξύ των αλγορίθμων BFS, DFS, A* (με Manhattan Distance ως Heuristic), A* (με Manhattan Distance ως Heuristic αλλα και Reversal Penalty), B&B και IDS και καταγράφονται εκατό εκτελέσεις. Για κάθε μία από τις εκατό εκτελέσεις παρουσιάζονται οι χρόνοι εκτέλεσης των έξι αλγορίθμων, ο αριθμός των κόμβων που χρειάστηκε να επεκταθούν καθώς και ο αριθμός των βημάτων που κάνει το πιόνι του Ιππότη μέχρι να φτάσει στην κατάσταση στόχο.

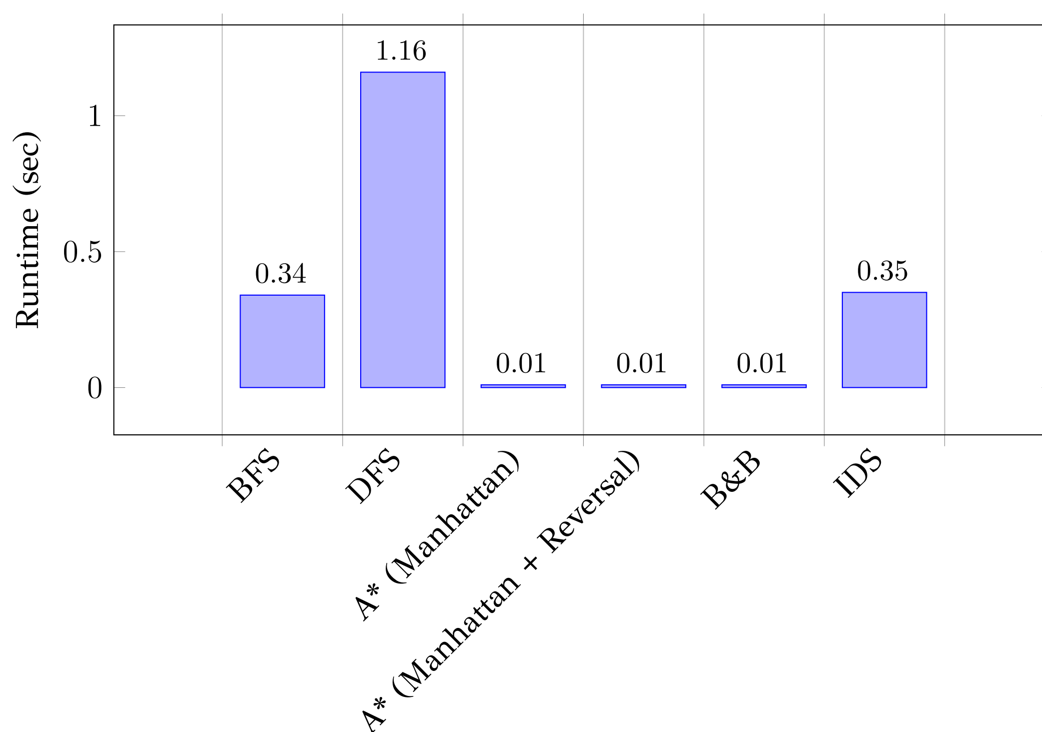
Ο Πίνακας 4.6 παρουσιάζει τον μέσο χρόνο εκτέλεσης αλλά και τον μέσο αριθμό κόμβων που επεκτάθηκαν, για κάθε έναν από τους έξι αλγορίθμους στις εκατό καταγεγραμμένες εκτελέσεις του Knight Problem.

Ομοίως, τα Σχήματα 4.7 και 4.8 απεικονίζουν τους μέσους όρους των χρόνων εκτέλεσης και αριθμών κόμβων, αντίστοιχα, που παρουσιάζονται στον Πίνακα 4.6.

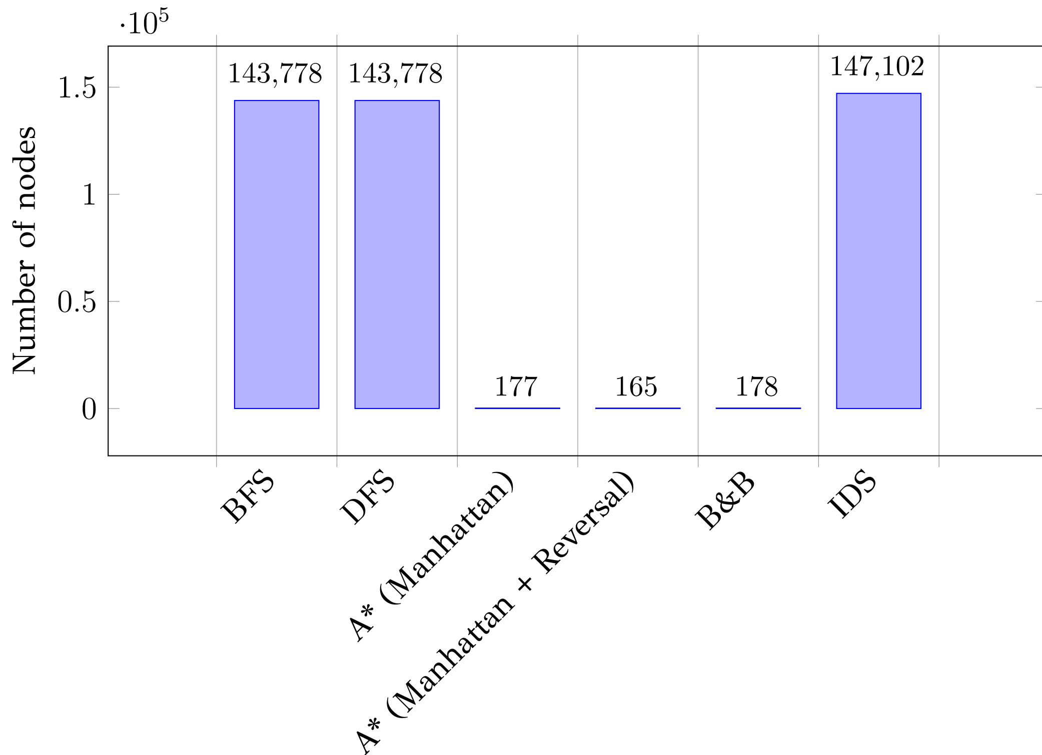
Αυτή τη φορά, παρατηρώντας τον Πίνακα 4.6 αλλά και τα Σχήματα 4.7 και 4.8 είναι εμφανές πως για το συγκεκριμένο πρόβλημα, από τους αλγορίθμους που δε χρησιμοποιούν κάποια ευρετική διαδικασία, ο BFS και ο IDS ευνοούνται σε σύγκριση

Πίνακας 4.6: Μέσοι όροι του Knight Problem.

Αλγόριθμος	**Χρόνος Εκτέλεσης**	**Αριθμός Κόμβων**
BFS	0.34	143,778
DFS	1.16	143,778
A* (Manhattan)	0.01	177
A* (Manhattan + Reversal)	0.01	165
B&B	0.01	178
IDS	0.35	147,102



Σχήμα 4.7: Μέσος όρος χρόνου εκτέλεσης από 100 μετρήσεις



Σχήμα 4.8: Μέσος όρος αριθμού κόμβων που επεκτάθηκαν από 100 μετρήσεις

με τον DFS, καθώς έχουν αρκετά μικρότερο χρόνο εκτέλεσης ενώ επεκτείνουν, αν όχι τον ίδιο, πολύ κοντινό αριθμό κόμβων. Ενώ, στην περίπτωση των αλγορίθμων που χρησιμοποιούν ευρετική διαδικασία, οι μετρήσεις είναι πολύ πιο κοντά μεταξύ τους παρουσιάζοντας πολύ μικρές αυξομειώσεις τόσο στον χρόνο εκτέλεσης όσο και στον αριθμό των κόμβων που επεκτάθηκαν. Όντας ένα πρόβλημα βέλτιστου μονοπατιού όπου η θέση στόχος είναι σε μία συγκεκριμένη κατεύθυνσή από το πιόνι κάθε φορά, παρατηρείται πως, σε σύγκριση με το 8-Puzzle, ο A* (Manhattan Distance plus Reversal Penalty) επεκτείνει, αν και με μικρή διαφορά, λιγότερους κόμβους από τον A* (Manhattan Distance) καθώς ευνοείται η διατήρηση της ίδιας κατεύθυνσης.

4.3 Συμπεράσματα πειραμάτων

Έπειτα από τις δοκιμές και τις συγκρίσεις των μετρήσεων που εξήχθησαν από τα πειράματα που διεξάχθηκαν, εύκολα συμπεραίνει κανείς πως ο Min-Max πραγματοποιεί ακριβώς τον στόχο που του τέθηκε και είναι αδύνατον να χάσει μία παρτίδα τρίλιζας. Για το Connect-4 ο Alpha-Beta Pruning υπερισχύει του Min-Max καθώς βρίσκει τα ίδια αποτελέσματα αλλά με καλύτερη επίδοση. Τέλος, για το 8-Puzzle προκύπτει ότι ο ιδανικότερος από τους αλγορίθμους που συγκρίθηκαν είναι ο A*

με απόσταση Manhattan ως ευρετική διαδικασία, ενώ για το Knight Problem, αν και η διαφορά μεταξύ τους ήταν μικρή, τον ιδανικότερο αλγόριθμο αποτέλεσε ο A* με απόσταση Manhattan και Reversal Penalty για ευρετική διαδικασία.

Κεφάλαιο 5

Συμπεράσματα

Οι έρευνες στον κλάδο των συνδυαστικών προβλημάτων ποικίλλουν, από προσπάθειες για καινούριες ανακαλύψεις, καινοτομίες βασισμένες σε στοιχεία προγενέστερων υλοποιήσεων, ακόμα και για αξιοποίηση υλοποιήσεων άλλου σκοπού πάνω σε πιο συγκεκριμένες έρευνες. Η υποκατηγορία των συνδυαστικών προβλημάτων που ασχολείται η συγκεκριμένη διπλωματική, τα συνδυαστικά παιχνίδια, δε διαφέρει. Σκοπός είναι να ξεχωρίσουν οι βέλτιστες ρυθμίσεις, ομαδοποιήσεις ή επιλογές στοιχείων, που συχνά αντιπροσωπεύουν την πιο συμφέρουσα κίνηση για έναν παίκτη σε κάθε δεδομένη στιγμή. Αυτός είναι και ο στόχος αυτής της διπλωματικής, στην οποία υλοποιούνται οι αλγόριθμοι Min-Max, Alpha-Beta Pruning, Breadth-First Search (BFS), Depth-First Search (DFS), A* (με Manhattan Distance ως Heuristic), A* (με Manhattan Distance ως Heuristic αλλα και Reversal Penalty), Branch-And-Bound Search (B&B) και Iterative-Deepening Search (IDS) με σκοπό να μελετηθούν τα αποτελέσματα τους και να ξεχωρίσει ο ιδανικότερος για το εκάστοτε πρόβλημα. Η έρευνα εκτυλίσσεται σε διάφορες διαστάσεις: μία αήττητη υλοποίηση του Min-Max για το ταμπλό της τρίλιζας, μια συγκριτική ανάλυση μεταξύ Min-Max και Alpha-Beta Pruning στο πλαίσιο του παιχνιδιού Connect-4, η λύση του 8-Puzzle από τους αλγορίθμους BFS, DFS, A* (Manhattan Distance), A* (Manhattan Distance plus Reversal Penalty), B&B και IDS, καθώς και η εύρεση του ιδανικότερου μονοπατιού για ένα πιόνι ιππότη σε μια σκακιέρα NxN, πάλι από τους BFS, DFS, A* (Manhattan Distance), A* (Manhattan Distance plus Reversal Penalty), B&B και IDS. Τα συμπεράσματα των μετρήσεων και συγκρίσεων οδήγησαν στα πορίσματα ότι όταν έρχονται αντιμέτωποι με τον Min-Max, οι χρήστες μπορούν να πετύχουν στην καλύτερη περίπτωση ισοπαλία, ο Alpha-Beta Pruning αναδεικνύεται ιδανι-

κότερος από τον Min-Max, καθώς παρέχει ισοδύναμα αποτελέσματα αλλά με πιο αποδοτικό, από άποψη χρόνου τρόπο, ο A* με Manhattan Distance για ευρετική διαδικασία είναι ο ιδανικότερος, συγκριτικά με τους άλλους για την επίλυση του 8-Puzzle και τέλος πως ο A* με Manhattan Distance για ευρετική διαδικασία και Reversal Penalty είναι ο ιδανικότερος, συγκριτικά με τους άλλους, για την εύρεση του συντομότερου μονοπατιού πάνω στη σκακιέρα $N \times N$. Μελλοντικές προσπάθειες θα μπορούσαν να συμπεριλαμβάνουν την ενσωμάτωση επιπρόσθετων συγκριτικών αλγορίθμων, την εξερεύνηση διαφορετικών ευρετικών διαδικασιών ή την εφαρμογή αυτών των αλγορίθμων σε ένα νέο φάσμα συνδυαστικών παιχνιδιών, δίνοντας τη δυνατότητα εξαγωγής νέων πορισμάτων από τις καινούργιες μετρήσεις.

Βιβλιογραφία

[1] Azmi Alazzam and Harold W Lewis. A new optimization algorithm for combinatorial problems. *IJARAI) International Journal of Advanced Research in Artificial Intelligence*, 2(5), 2013.

[2] Anurag Bhatt, Pratul Varshney, and Kalyanmoy Deb. In search of no-loss strategies for the game of tic-tac-toe using a customized genetic algorithm. In *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation*, pages 889–896, 2008.

[3] Kenneth J Chisholm and Peter VG Bradbeer. Machine learning using a genetic algorithm to optimise a draughts program board evaluation function. In *Proceedings of 1997 IEEE International Conference on Evolutionary Computation (ICEC'97)*, pages 715–720. IEEE, 1997.

[4] Colin Clausen, Simon Reichhuber, Ingo Thomsen, and Sven Tomforde. Improvements to increase the efficiency of the alphazero algorithm: A case study in the game'connect 4'. In *ICAART (2)*, pages 803–811, 2021.

[5] Erik D Demaine. Playing games with algorithms: Algorithmic combinatorial game theory. In *Mathematical Foundations of Computer Science 2001: 26th International Symposium, MFCS 2001 Mariánské Lázne, Czech Republic, August 27–31, 2001 Proceedings 26*, pages 18–33. Springer, 2001.

[6] Aviezri Fraenkel. Combinatorial games: selected bibliography with a succinct gourmet introduction. *The Electronic Journal of Combinatorics*, pages DS2–Aug, 2012.

[7] Richard K Guy. Unsolved problems in combinatorial games. *Combinatorics Advances*, pages 161–179, 1995.

[8] Ami Hauptman and Moshe Sipper. Evolution of an efficient search algorithm for the mate-in-n problem in chess. In *Genetic Programming: 10th European Conference, EuroGP 2007, Valencia, Spain, April 11-13, 2007. Proceedings 10*, pages 78–89. Springer, 2007.

[9] Brian Hegerty, Chih-Cheng Hung, and Kristen Kasprak. A comparative study on differential evolution and genetic algorithms for some combinatorial problems. In *Proceedings of 8th Mexican International Conference on Artificial Intelligence*, volume 9, page 13, 2009.

[10] Gregor Hochmuth. On the genetic evolution of a perfect tic-tac-toe strategy. *Genetic Algorithms and Genetic Programming at Stanford*, pages 75–82, 2003.

[11] Anca-Elena Iordan. A comparative study of three heuristic functions used to solve the 8-puzzle. *Br. J. Math. Comput. Sci*, 16(1):1–18, 2016.

[12] David S Johnson. Approximation algorithms for combinatorial problems. In *Proceedings of the fifth annual ACM symposium on Theory of computing*, pages 38–49, 1973.

[13] Daniel R Kunkle. Solving the 8 puzzle in a minimum number of moves: An application of the a* algorithm. *Introduction to Artificial Intelligence*, 2001.

[14] Rijul Nasa, Rishabh Didwania, Shubhranil Maji, and Vipul Kumar. Alpha-beta pruning in mini-max algorithm–an optimized approach for a connect-4 game. *Int. Res. J. Eng. Technol*, pages 1637–1641, 2018.

[15] Karmanya Oberoi, Sarthak Tandon, Abhishek Das, and Swati Aggarwal. An evolutionary approach to combinatorial gameplaying using extended classifier systems. In *Applications of Artificial Intelligence and Machine Learning: Select Proceedings of ICAAAIML 2020*, pages 723–738. Springer, 2021.

[16] Alexander Reinefeld. Complete solution of the eight-puzzle and the bene t of node ordering in ida*. In *International Joint Conference on Artificial Intelligence*, pages 248–253. Citeseer, 1993.

[17] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

[18] Thomas Stützle. Local search algorithms for combinatorial problems. *Darmstadt University of Technology PhD Thesis*, 20, 1998.

[19] Thomas Thomsen. Lambda-search in game trees—with application to go. In *Computers and Games: Second International Conference, CG 2000 Hamamatsu, Japan, October 26–28, 2000 Revised Papers 2*, pages 19–38. Springer, 2001.

[20] Jos WHM Uiterwijk. Solving cram using combinatorial game theory. In *Advances in Computer Games: 16th International Conference, ACG 2019, Macao, China, August 11–13, 2019, Revised Selected Papers*, pages 91–105. Springer, 2020.

[21] Jos WHM Uiterwijk and Janis Griebel. Combining combinatorial game theory with an-solver for clobber: Theory and experiments. In *BNAIC 2016: Artificial Intelligence: 28th Benelux Conference on Artificial Intelligence, Amsterdam, The Netherlands, November 10-11, 2016, Revised Selected Papers*, pages 78–92. Springer, 2017.

[22] Shi-Jim Yen and Jung-Kuei Yang. Two-stage monte carlo tree search for connect6. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(2):100–118, 2011.