Εργασία Τεχνητής Νοημοσύνης 2020

Σπύρος Μαντέλος Α.Μ: 4104 (cs04104@uoi.gr)
Χρήστος Γεωργίου Μουσσές Α.Μ: 4206 (cs04206@uoi.gr)
Γεώργιος Μήτσης Α.Μ: 4258 (cs04258@uoi.gr)

Άσκηση 1

Στην πρώτη άσκηση υλοποιήθηκαν οι αλγόριθμοι **UCS** και **A*** στα πλαίσια του προβλήματος μετακινήσεων σφαιρών. Το πρόβλημα απαιτεί οι αρχικές καταστάσεις να περιέχουν ίσο αριθμό Ν από μαύρες (Μ) και άσπρες (Α) σφαίρες, και μία κενή θέση (-). Στη συνέχεια, ανταλλάσοντας σε κάθε βήμα μία οποιαδήποτε σφαίρα με την κενή θέση, με τον περιορισμό ότι η απόσταση μεταξύ τους να είναι το πολύ ίση με Ν και το κόστος γι' αυτή τη μεταφορά να ισούται με την απόσταση της μετακινούμενης σφαίρας από την κενή θέση, θέλουμε να φτάσουμε σε μία τελική κατάσταση όπου όλες οι μαύρες σφαίρες βρίσκονται σε θέσεις αριστερά από τις λευκές, ενώ στην δεξιότερη θέση βρίσκεται υποχρεωτικά μια λευκή σφαίρα.

Αλγόριθμος

Ο γενικός αλγόριθμος που ακολουθήθηκε είναι ο γενικός αλγόριθμος αναζήτησης που περιγράφεται στις διαφάνειες του μαθήματος, ο οποίος περιλαμβάνει τα ακόλουθα βήματα:

1. Βάλε την αρχική κατάσταση στο μέτωπο της αναζήτησης.
2. Αν το μέτωπο αναζήτησης είναι κενό τότε τερμάτισε τον αλγόριθμο.
3. Πάρε την πρώτη σε σειρά κατάσταση του μετώπου της αναζήτησης.
4. Αν αυτή η κατάσταση ανήκει στο κλειστό σύνολο, τότε αφαίρεσέ τη από το μέτωπο αναζήτησης και επίστρεψε στο βήμα 2.
5. Αν η κατάσταση αυτή είναι τελική, τότε τύπωσε τη λύση και τερμάτισε τον αλγόριθμο.
6. Εφάρμοσε τους τελεστές μετάβασης για να παράγεις τις καταστάσεις-παιδιά.
7. Βάλε τις νέες καταστάσεις-παιδιά στο μέτωπο της αναζήτησης (σύμφωνα με το κριτήριο που χαρακτηρίζει τον αλγόριθμο).
8. Βάλε την κατάσταση-γονέα στο κλειστό σύνολο.
9. Επίστρεψε στο βήμα 2.

Για τον αλγόριθμο **UCS**, στο βήμα 3 επιλέγεται η κατάσταση με το μικρότερο κόστος από την αφετηρία μέχρι την τρέχουσα κατάσταση. Το κόστος υπολογίζεται κατά την επέκταση σε κάθε παιδί και ισούται με το κόστος της γονικής κατάστασης συν το κόστος μετάβασης από τη γονική κατάσταση στο εκάστοτε παιδί. Έτσι ο αλγόριθμος συνεχίζει μέχρι να βρει μία τελική κατάσταση με μία βέλτιστη, ως προς το συνολικό κόστος, ακολουθία κινήσεων.

Ευρετική Συνάρτηση

Για να προκύψει βέλτιστη λύση από τον αλγόριθμο **A***, πρέπει να χρησιμοποιήσουμε μία αποδεκτή ευρετική συνάρτηση. Για να είναι αποδεκτή, πρέπει η εκτίμηση της απόστασης από την τελική κατάσταση τη οποία επιστρέφει η ευρετική συνάρτηση, να είναι μικρότερη ή ίση από την πραγματική απόσταση, για όλες τις καταστάσεις.

Μια στρατηγική κατασκευής αποδεκτών ευρετικών συναρτήσεων είναι, αφαιρώντας περιορισμούς, να κατασκευάσουμε ένα χαλαρωμένο πρόβλημα στο οποίο μπορούμε να βρούμε τη βέλτιστη λύση η οποία θα είναι ένας αποδεκτός ευρετικός μηχανισμός για το αρχικό μας πρόβλημα.

Ως χαλαρωμένο πρόβλημα θεωρήσαμε το πρόβλημα το οποίο προκύπτει μετά την άρση του περιορισμού ανταλλαγής της κενής θέσης με σφαίρα μόνο μέχρι N θέσεις μακριά (όπου N ο αριθμός των λευκών ή μαύρων σφαιρών) και την άρση του περιορισμού για ύπαρξη λευκής σφαίρας στην τελευταία δεξιά θέση στην τελική κατάσταση.

Ορίζουμε ως εσφαλμένα τοποθετημένες σφαίρες, τις λευκές σφαίρες που βρίσκονται αριστερά του κέντρου και τις μαύρες σφαίρες που βρίσκονται δεξιά του κέντρου όπου το κέντρο συμπίπτει με το σύμβολο στη θέση N + 1.

Αρχικά χρησιμοποιήσαμε ως τιμή της ευρετικής συνάρτησης, το άθροισμα των αποστάσεων όλων των εσφαλμένα τοποθετημένων σφαιρών από το κέντρο, το οποίο αποτελεί μια εκτίμηση του πόσο μακριά βρισκόμαστε από μια διαμόρφωση με τις μαύρες σφαίρες να βρίσκονται αριστερά και τις λευκές δεξιά των μαύρων.

Πράγματι, οι εσφαλμένα τοποθετημένες μαύρες σφαίρες πρέπει να μετακινηθούν αριστερά του κέντρου και οι εσφαλμένα λευκές σφαίρες δεξιά του κέντρου. Στην πράξη όμως, αποδείχθηκε πως αυτή είναι μια αρκετά συντηρητική εκτίμηση της πραγματικής απόστασης, αφενός επειδή δεν τοποθετεί τις σφαίρες στην τελική τους θέση, και αφετέρου δε λαμβάνει υπ' όψη οτι οι ανταλλαγές πρέπει να γίνονται μέσω της κενής θέσης.

Έτσι, οδηγηθήκαμε στο να υπολογίζουμε κάθε φορά τη συνολική απόσταση που θα διανύσει η κενή θέση κάνοντας εναλασσόμενες κινήσεις αριστερά και δεξιά

του κέντρου, κινούμενη κάθε φορά μόνο προς εσφαλμένα τοποθετημένες σφαίρες. Με τον τρόπο αυτό, σε κάθε κίνηση της κενής θέσης, μια εσφαλμένα τοποθετημένη σφαίρα τοποθετείται σε σωστή θέση και έχουμε επίσης τον ελάχιστο αριθμό κινήσεων για να φέρουμε την αρχική διαμόρφωση σε μια αποδεκτή τελική διαμόρφωση, σύμφωνα με τις συνθήκες του χαλαρωμένου προβλήματος.

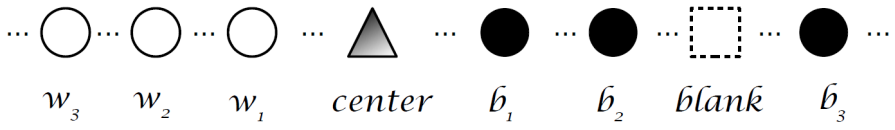
Πρέπει επίσης να μειώσουμε το κόστος στο ελάχιστο, με σκοπό να βρούμε μια βέλτιστη λύση για το χαλαρωμένο πρόβλημα η οποία θα αποτελέσει αποδεκτή ευρετική συνάρτηση που υποεκτιμά το πραγματικό κόστος του αρχικού προβλήματος, αλλά με όσο μικρότερη διαφορά γίνεται. Γι' αυτό αναλύσαμε τις διαφορετικές διαμορφώσεις που μπορεί να συναντήσουμε.

Θεωρώντας το γενικό πλαίσιο, ας υποθέσουμε οτι η κενή θέση βρίσκεται στα δεξιά του κέντρου, και απέχει απόσταση κ από αυτό και πως υπάρχει μια εσφαλμένα τοποθετημένη λευκή σφαίρα καθώς και μια εσφαλμένα τοποθετημένη μαύρη σφαίρα. Σύμφωνα με τον τρόπο μετακινήσεων με εναλλαγές κατεύθυνσης που υιοθετούμε, η κενή θέση θα κινηθεί αρχικά προς τα αριστερά προς τη λευκή σφαίρα, η οποία απέχει w από το κέντρο και στη συνέχεια θα κινηθεί προς τα δεξιά, προς τη μαύρη σφαίρα, η οποία απέχει b από το κέντρο. Έτσι, το συνολικό κόστος για τις δύο αυτές κινήσεις θα είναι:

$$(\kappa + w) + (w + b) = \kappa + (w + b) + w$$

Για να ελαχιστοποιηθεί το κόστος αυτό, πρέπει αφενός να ελαχιστοποιηθεί το $w + b$, δηλαδή η απόσταση μεταξύ των 2 σφαιρών, και αφετέρου το w. Αυτό επιτυγχάνεται συγχρόνως, εάν οι δύο σφαίρες είναι όσο το δυνατόν πιο κοντά στο κέντρο (από τα αριστερά και τα δεξιά του, αντίστοιχα). Δηλαδή, για να προκύψει το μικρότερο κόστος μετακίνησης, θα πρέπει η κενή θέση να κινείται εναλλάξ προς τις κοντινότερες στο κέντρο λανθασμένα τοποθετημένες σφαίρες.

Έτσι, στο παράδειγμα της εικόνας 1, θα έχουμε συνολικά τη σειρά μετατοπίσεων $blank \rightarrow w_1 \rightarrow b_1 \rightarrow w_2 \rightarrow b_2 \rightarrow w_3 \rightarrow b_3 \rightarrow ...$ όπου w_1, b_1 οι εγγύτερες στο κέντρο λανθασμένα τοποθετημένες άσπρη και μαύρη σφαίρες (εδώ ονομάσαμε τις σφαίρες από τις αποστάσεις τους από το κέντρο).



Εικόνα 1: Παράδειγμα τοποθέτησης σφαιρών

Χρειάζεται όμως να διευκρινίσουμε περαιτέρω τι πρέπει να πράξουμε σε σχέση με αυτές τις εναλλαγές, στην περίπτωση που αρχικά η κενή θέση βρίσκεται στο κέντρο. Επίσης, θα πρέπει να αποφασίσουμε, στην περίπτωση όπου μια λευκή, ή μια μαύρη σφαίρα βρίσκεται στην κεντρική θέση, αν θα τη θεωρήσουμε ως εσφαλμένα τοποθετημένη, ή όχι. Συγκεκριμένα, διακρίνουμε πέντε διαφορετικές περιπτώσεις τις οποίες αναλύουμε στη συνέχεια.

Στην πρώτη περίπτωση (εικόνα 2), όπου η κενή θέση βρίσκεται στο κέντρο, αποδεικνύεται εύκολα πως όσες εσφαλμένα τοποθετημένες σφαίρες έχουμε αριστερά της κενής θέσης, τόσες εσφαλμένα τοποθετημένες μαύρες σφαίρες έχουμε και δεξιά της κενής θέσης. Άν έχουμε k λευκές και k μαύρες εσφαλμένα τοποθετημένες σφαίρες, το συνολικό κόστος, άν ξεκινήσουμε τις εναλλαγές από αριστερά είναι:

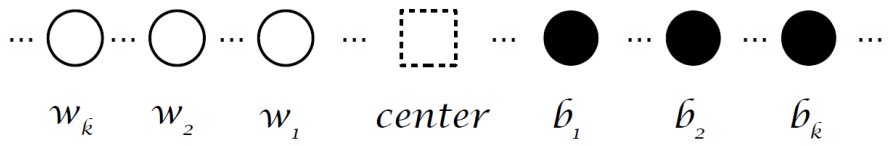
$$cost_{left} = w_1 + (w_1 + b_1) + (b_1 + w_2) + (w_2 + b_2) + ... + (b_{k-1} + w_k) + (w_k + b_k)$$

ενώ αν ξεκινήσουμε από τα δεξιά, το κόστος είναι:

$$cost_{right} = b_1 + (b_1 + w_1) + (w_1 + b_2) + (b_2 + w_2) + ... + (w_{k-1} + b_k) + (b_k + w_k)$$

Άν συγκρίνουμε τις 2 ποσότητες, παρατηρούμε πως εν τέλει έχουμε $cost_{left} = A + b_k$ και $cost_{right} = A + w_k$, όπου A εύκολα υπολογίσιμη ποσότητα.

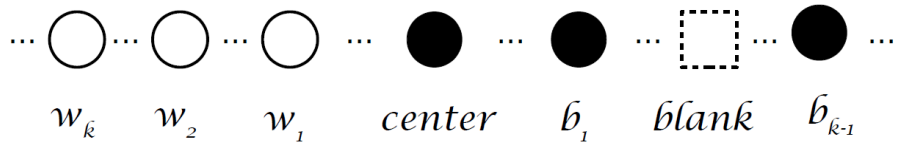
Άρα, για να ελαχιστοποιήσουμε το κόστος, αρκεί να επιλέξουμε να ξεκινήσουμε τις εναλλαγές από την πλευρά στην οποία ανήκει το $min(w_k, b_k)$.



Εικόνα 2: Περίπτωση όπου η κενή θέση βρίσκεται στην κεντρική θέση

Στις υπόλοιπες τέσσερεις περιπτώσεις που χρειάζεται να αναλυθούν, θα έχουμε στο κέντρο είτε λευκή είτε μαύρη σφαίρα και η κενή θέση θα βρίσκεται αρχικά είτε δεξιά είτε αριστερά από το κέντρο.

Θα αναλύσουμε το εάν για να προκύψει μικρότερο κόστος, πρέπει να συμπεριλάβουμε και τη εκάστοτε σφαίρα που βρίσκεται στην κεντρική θέση ως εσφαλμένα τοποθετημένη. Από τις τέσσερεις περιπτώσεις, αναλύουμε αρχικά την περίπτωση όπου υπάρχει μαύρη σφαίρα στην κεντρική θέση, και η κενή θέση βρίσκεται στα δεξιά της κεντρικής. Σε αυτή την περίπτωση αποδεικνύεται εύκολα πως οι εσφαλμένα τοποθετημένες λευκές σφαίρες αριστερά του κέντρου είναι κατα μία περισσότερες από τις εσφαλμένα τοποθετημένες μαύρες σφαίρες δεξιά του κέντρου.



Εικόνα 3: Περίπτωση όπου η κενή θέση βρίσκεται στα δεξιά της κεντρικής και στην κεντρική θέση υπάρχει μαύρη σφαίρα

Άν δε λάβουμε υπ' όψη ως εσφαλμένα τοποθετημένη την κεντρική μαύρη σφαίρα, οι εναλλαγές έχουν ως εξής:

$$blank \to w_1 \to b_1 \to w_2 \to b_2 \to ... \to w_{k-1} \to b_{k-1} \to w_k$$

με κόστος:

$$cost_{without} = (\kappa+w_1)+(w_1+b_1)+(b_1+w_2)+(w_2+b_2)+...+(w_{k-1}+b_{k-1})+(b_{k-1}+w_k)$$

Αν λάβουμε υπ' όψη ως εσφαλμένα τοποθετημένη την κεντρική μαύρη σφαίρα, οι εναλλαγές έχουν ως εξής:

$$blank \to w_1 \to center \to w_2 \to b_1 \to w_3 \to b_2 \to ... \to w_{k-1} \to b_{k-2} \to w_k \to b_{k-1}$$

με κόστος:

$$cost_{with} = (\kappa+w_1)+w_1+w_2+(w_2+b_1)+(b_1+w_3)+...+(b_{k-2}+w_k)+(w_k+b_{k-1})$$

Άν συγκρίνουμε τις 2 ποσότητες παρατηρούμε, πως εν τέλει έχουμε $cost_{without} = A + b_{k-1}$ και $cost_{with} = A + w_k$, όπου A εύκολα υπολογίσιμη ποσότητα.

Για να ελαχιστοποιήσουμε το κόστος στην περίπτωση αυτή, όπου η κεντρική σφαίρα είναι μαύρη και η κενή θέση είναι δεξιά της, πρέπει να θεωρήσουμε την κεντρική σφαίρα ως εσφαλμένη μόνο στην περίπτωση όπου $w_k < b_{k-1}$.

Με παρόμοιο τρόπο αποδεικνύονται οι συνθήκες κάτω από τις οποίες η κεντρική σφαίρα πρέπει να θεωρηθεί εσφαλμένα τοποθετημένη με σκοπό την ελαχιστοποίηση του κόστους και οι οποίες παρατίθενται στις γραμμές 2 - 4 του παρακάτω πίνακα. Να διευκρινίσουμε οτι η τοποθέτηση (left/right) της κενής θέσης στον παρακάτω πίνακα γίνεται ως προς τη κεντρική θέση και πως με w_0 ή b_0 συμβολίζουμε την κεντρική λευκή η μαύρη σφαίρα αντίστοιχα, άν αυτή πρέπει να θεωρηθεί ως εσφαλμένα τοποθετημένη.

center symbol	blank position	# of wrongly placed whites	# of wrongly placed blacks	L	R	starting from	condition
-	center	k	k	w_1 w_2 ... w_k	b_1 b_2 ... b_k	L	$w_k < b_k$
-	center	k	k	w_1 w_2 ... w_k	b_1 b_2 ... b_k	R	$b_k < w_k$
M	right	k	k-1	w_1 w_2 ... w_k	b_0 b_1 ... b_{k-1}	L	$w_k < b_{k-1}$
M	left	k	k	w_1 w_2 ... w_k	b_0 b_1 ... b_k	R	$w_k < b_k$
A	right	k	k	w_0 w_1 ... w_k	b_1 b_2 ... b_k	L	$b_k < w_k$
A	left	k-1	k	w_0 w_1 ... w_{k-1}	b_1 b_2 ... b_k	R	$b_k < w_{k-1}$

Πίνακας 1: Περιπτώσεις, είτε με την κενή θέση στο κέντρο, είτε για τη θεώρηση κεντρικά τοποθετημένης λευκής ή μαύρης σφαίρας ως εσφαλμένα τοποθετημένης

Για καθεμία από τις περιπτώσεις του παραπάνω πίνακα που αναλύσαμε, έχουμε προσδιορίσει μια βέλτιστη αλληλουχία κινήσεων που δίνει το μικρότερο κόστος. Πρόκειται για την τύπου πέρα-δώθε αλληλουχία κινήσεων, όπου κινούμαστε κάθε φορά προς την πιο κοντινή στο κέντρο εσφαλμένα τοποθετημένη σφαίρα, η οποία βρίσκεται στην αντίθετη προς το κέντρο, πλευρά από αυτή της κενής θέσης. Η αλληλουχία αυτή έχει, για το χαλαρωμένο πρόβλημα, μικρότερο κόστος από οποιαδήποτε άλλη αλληλουχία η οποία περιλαμβάνει και μεταβάσεις σε λευκές ή μαύρες σφαίρες οι οποίες είναι σωστά τοποθετημένες. Πράγματι η αλληλουχία είναι βέλτιστη μέχρι να επιλεγεί η κίνηση προς κάποια σωστά τοποθετημένη σφαίρα και παραμένει βέλτιστη από τη διαμόρφωση που προκύπτει μετά την κίνηση αυτή και μέχρι να προσεγγίσουμε την τελική κατάσταση. Η μετάβαση προς κάποια σωστά τοποθετημένη σφαίρα, απλά δημιουργεί επιπρόσθετο κόστος, χωρίς προσδοκία βελτίωσης του συνολικού κόστους από τις επόμενες από αυτήν κινήσεις.

Να αναφέρουμε τέλος, οτι σκεφτήκαμε πως το χαλαρωμένο πρόβλημα μπορεί να αναπαρασταθεί με ένα διμερές γράφημα, όπου στο ένα σύνολο έχουμε τις λευκές εσφαλμένα τοποθετημένες σφαίρες και στο άλλο τις μαύρες. Η κεντρική άσπρη, ή μαύρη σφαίρα, όταν λαμβάνεται υπ' όψη ως εσφαλμένα τοποθετημένη, περιλαμβάνεται στο σύνολο του χρώματός της. Η κενή θέση προστίθεται, είτε στο χρώμα της αντίθετης περιοχής στην οποία βρίσκεται αρχικά (δεξιά-λευκό αριστερά-μαύρο), είτε στο χρώμα της αντίθετης περιοχής προς την οποία ξεκινάμε τις κινήσεις, άν είναι τοποθετημένη στο κέντρο. Τότε, το πρόβλημά μας ανάγεται στο να βρούμε μια ελάχιστη διαδρομή Hamilton που περνάει ακριβώς μία φορά από κάθε κορυφή του γραφήματος, ξεκινώντας από την κενή θέση, εναλασσόμενη κάθε φορά μεταξύ των δύο συνόλων του γραφήματος.

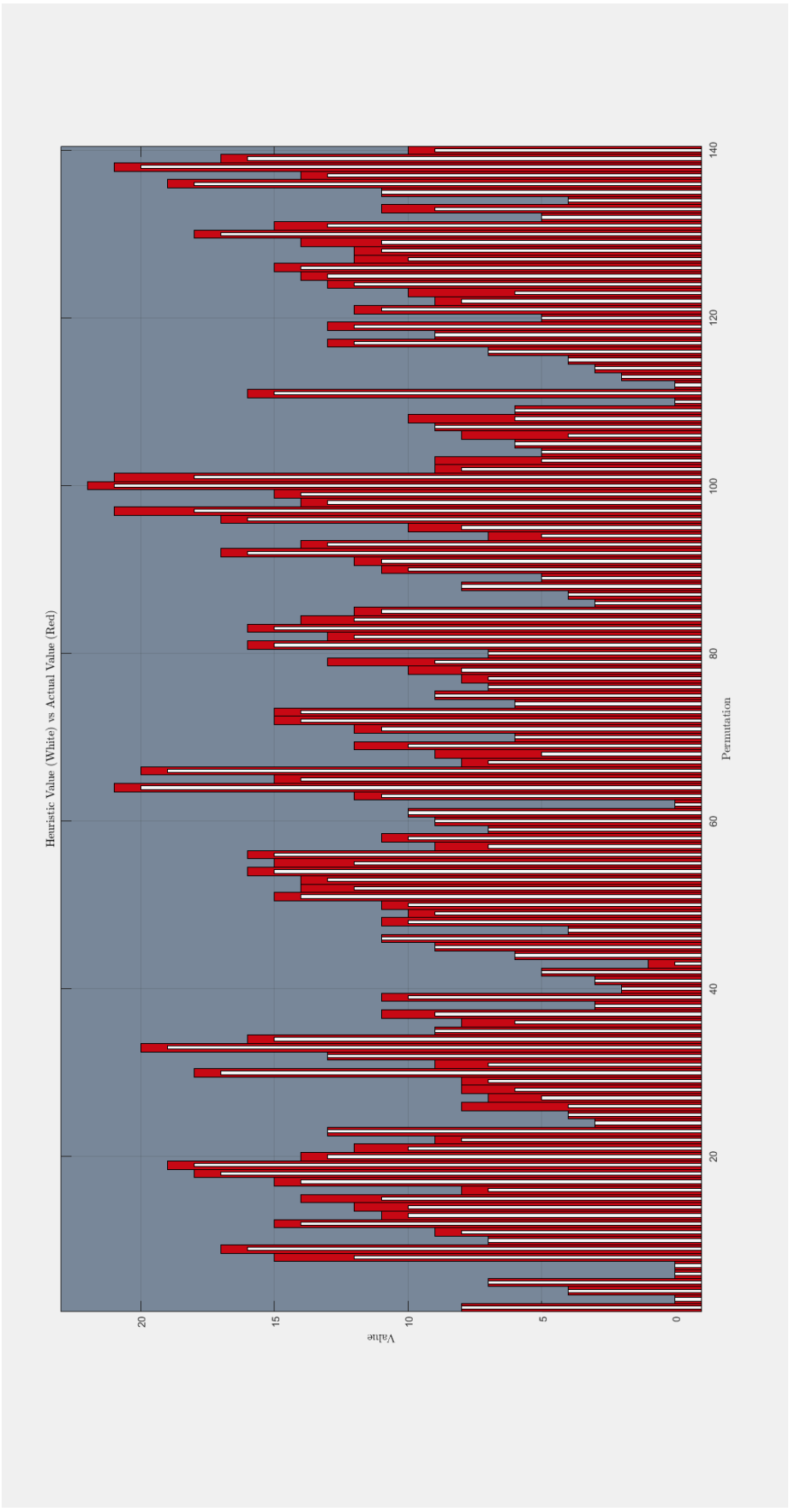
Ελέγξαμε το άν οι τιμές της ευρετικής συνάρτησης αποτελούν υποεκτίμηση του πραγματικού κόστους για όλους τους συνδυασμούς σφαιρών από $N = 1$ έως $N = 6$. Όταν έχουμε N μαύρες και N άσπρες σφαίρες, όλες οι διαφορετικές αρχικές καταστάσεις είναι $(2N+1)\begin{pmatrix} 2N \\ N \end{pmatrix}$, δηλαδή $\begin{pmatrix} 2N \\ N \end{pmatrix}$ διαφορετικές τοποθετήσεις N λευκών σφαιρών και N μαύρων σφαιρών για καθεμία από τις (2N + 1) θέσεις τοποθέτησης της κενής θέσης, όπως φαίνεται και στον παρακάτω πίνακα.

number of white/black balls	number of permutations
$N = 1$	6
$N = 2$	30
$N = 3$	140
$N = 4$	630
$N = 5$	2772
$N = 6$	12012

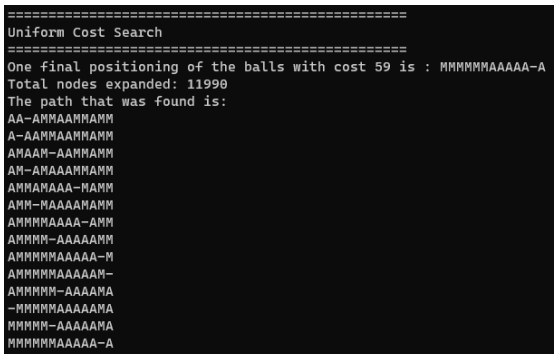
Εκτελέσαμε τον αλγόριθμο **UCS** για να λάβουμε το βέλτιστο κόστος για καθεμία από τις καταστάσεις του παραπάνω πίνακα και τις συγκρίναμε με την τιμή της ευρετικής μας συνάρτησης. Ενδεικτικά στην εικόνα 4 (σελίδα 9) φαίνονται τα αποτελέσματα για την περίπτωση $N = 3$, όπου είναι εμφανές πως η ευρετική μας

συνάρτηση, είτε βρίσκει το ελάχιστο κόστος σε κάποιες περιπτώσεις, είτε κάνει υποεκτίμησή του στις περισσότερες.

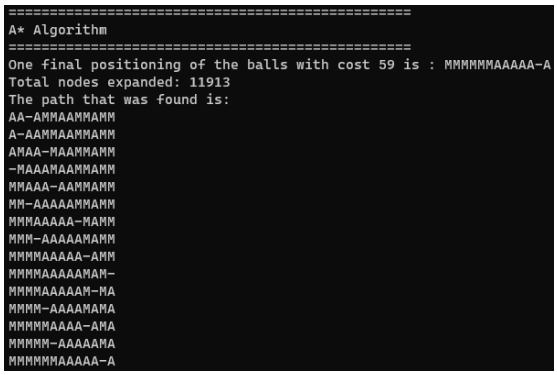
Συγκρίναμε επίσης, στην περίπτωση όπου για παράδειγμα η αρχική κατάσταση είναι **AA-AMMAAMMAMM**, τους αλγόριθμους **UCS**, **A*** με την αρχική συντηρητική ευρετική συνάρτηση και **A*** με την τελική επιλογή που κάναμε για την ευρετική συνάρτηση. Ο **UCS** χρειάστηκε περίπου 46 δευτερόλεπτα για να βρει τη βέλτιστη λύση, ο **A*** με την αρχική συντηρητική ευρετική συνάρτηση χρειάστηκε περίπου 53 δευτερόλεπτα και ο **A*** με την τελική επιλογή της ευρετικής συνάρτησης χρειάστηκε μόλις 4 δευτερόλεπτα. Επειδή οι χρόνοι δεν είναι ανεξάρτητοι από το άν έχει βελτιστοποιηθεί ο κώδικας που υλοποιεί τον αλγόριθμο, είναι πιο ενδεικτικό να συγκρίνουμε τους κόμβους που χρειάστηκε να επεκτείνει ο κάθε αλγόριθμος για να βρει τη βέλτιστη λύση. Ο **UCS** εξέτασε στο παραπάνω παράδειγμα 11990 κόμβους, ο **A*** με τη συντηρητική ευρετική συνάρτηση 11913 κόμβους και ο **A*** με την τελική μας επιλογή μόλις 4647 κόμβους. Επειδή η εκτίμηση του αλγόριθμου **A*** με την αρχικά επιλεγμένη ευρετική συνάρτηση είναι πολύ συντηρητική, ο αριθμός των κόμβων που επισκέπτεται πλησιάζει αρκετά κοντά αυτόν του **UCS** και ο **A*** με τη συντηρητική ευρετική συνάρτηση μπορεί σε κάποιες περιπτώσεις να χρειαστεί και περισσότερο χρόνο από τον **UCS** (λόγω του απαιτούμενου χρόνου για τον υπολογισμό και της ευρετικής τιμής). Οι λύσεις που βρήκαν και οι τρεις αλγόριθμοι είχαν κόστος 59 και όλοι οι αλγόριθμοι έχουν βρει διαφορετικές λύσεις, ενώ ο αλγόριθμος με την επιθετική ευρετική συνάρτηση που τελικά υιοθετήσαμε έχει βρει και λύση με μικρότερο αριθμό κινήσεων από τους άλλους δύο.



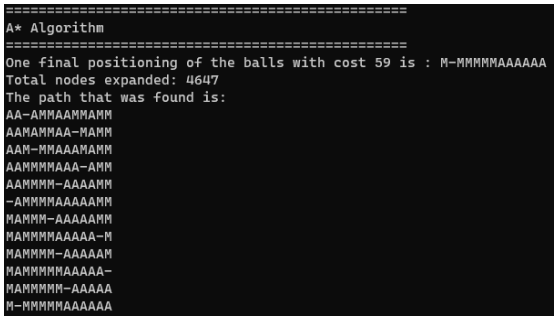
Εικόνα 4: Πραγματικό κόστος (κόκκινο) και εκτίμησή του (λευκό) για $N = 3$



Εικόνα 5: Λύση του **UCS**



Εικόνα 6: Λύση του **A*** με τη συντηρητική ευρετική συνάρτηση



Εικόνα 7: Λύση του **A*** με την επιθετική ευρετική συνάρτηση

Το γεγονός πως η επιθετική εκδοχή του **A*** χρειάζεται να εξετάσει αισθητά λιγότερους κόμβους για να βρει τη λύση, μας δίνει τη δυνατότητα να τον εφαρμόσουμε και σε συμβολοσειρές μεγαλύτερου μήκους. Για παράδειγμα, για την αρχική κατάσταση **AA-MMAAMAMMAMMMAA**, με $N = 8$ (συνολικά 17 σύμβολα), ο **A*** με την επιθετική ευρετική συνάρτηση, βρήκε λύση, εξετάζοντας 7970 κόμβους, σε 29.97 δευτερόλεπτα, ενώ ο **A*** με τη συντηρητική ευρετική συνάρτηση βρήκε λύση, εξετάζοντας 185.173 κόμβους σε 5.99 ώρες και ο **UCS** βρήκε λύση, εξετάζοντας 218.750 κόμβους σε 6.21 ώρες!

Λόγω του μεγάλου αριθμού των κόμβων, σε μεγάλες συμβολοσειρές, είναι απαραίτητο να αυξήσουμε το μέγεθος της μνήμης του **JVM** κατά την εκτέλεση του προγράμματος έτσι ώστε να μη γίνει υπερχείλιση στη μνήμη και το πρόγραμμα να σταματήσει να λειτουργει. Αυτό γίνεται μέσω της εντολής **java -Xms4096M filename.java**. Η παραπάνω εντολή, αυξάνει το μέγεθος της διαθέσιμης μνήμης του **heap** σε 4096mb αλλά ο χρήστης έχει τη δυνατότητα να αλλάξει το μέγεθός της σε κάποια επιθυμητή τιμή.

Η υλοποίηση του αλγόριθμου της επιθετικής ευρετικής συνάρτησης έγινε ως εξής: Χρησιμοποιούμε δύο **ArrayLists** (**left**, **right**) όπου τοποθετούμε αντίστοιχα τις θέσεις (με τιμές από 0 έως 2N + 1) των μαύρων και λευκών εσφαλμένα τοποθετημένων σφαιρών. Στη συνέχεια ανάλογα με την περίπτωση του πίνακα 1 στην οποία εντάσσεται η αρχική μας κατάσταση, προσθέτουμε, άν ισχύουν οι περιγραφόμενες στον πίνακα συνθήκες, είτε στον **left** είτε στον **right**, τη θέση της κεντρικής σφαίρας. Έπειτα, με τη χρήση μιας υπορουτίνας συγχώνευσης, συγχωνεύουμε εναλλάξ τα στοιχεία των δύο **ArrayList** (ανάλογα πάλι με την περίπτωση του πίνακα 1) και υπολογίζουμε από το νέο **ArrayList** που προκύπτει, την απόλυτη τιμή των διαφορών των διαδοχικών στοιχείων ανα δύο, τις οποίες και προσθέτουμε για να προκύψει η εκτίμηση του κόστους με βάση την ευρετική συνάρτηση.

	0	1	2	3	4	5	6	7	8
Έτσι για παράδειγμα, για την κατάσταση	M	A	A	M	M	A	A	-	M
		w_2	w_1						b_1

έχουμε:

left	right	blank	center
2	8	7	4
1			

Επειδή η κενή θέση είναι δεξιά του κέντρου ($7 > 4$), έχουμε μαύρη σφαίρα στο κέντρο και $w_2 = 4 - 1 = 3 < b_1 = 8 - 4 = 4$, λαμβάνουμε υπ' όψη την κεντρική μαύρη σφαίρα ως εσφαλμένα τοποθετημένη και τροποποιούμε το **ArrayList right** τοποθετώντας το δείκτη της κεντρικής μαύρης σφαίρας (4) ως πρώτο στοιχείο του:

left	right	blank
2	4	7
1	8	

'Ετσι, καταλήγουμε μετά την προσθήκη της κενής θέσης και των συγχωνευμένων (ξεκινώντας από αριστερά) **left** και **right** με την τελική μας λίστα να είναι τώρα:

merged
7
2
4
1
8

και με τελικό κόστος $cost = |7-2|+|2-4|+|4-1|+|1-8| = 5+2+3+7 = 17$

Υλοποίηση

Η υλοποίηση των αλγορίθμων, έγινε σε γλώσσα **Java**. Αρχικά ζητείται από το χρήστη να εισάγει την αρχική κατάσταση του προβλήματος. Εδώ του δίνονται δύο δυνατότητες: μπορεί είτε να εισάγει μια αρχική κατάσταση (η οποία ελέγχεται ώστε να είναι συμβατή με τις απαιτήσεις του προβλήματος), είτε να δώσει έναν αριθμό για το πλήθος των μπαλών του κάθε χρώματος (έστω N), ώστε να δημιουργηθεί τυχαία μία αρχική κατάσταση μήκους 2N+1. Στην συνέχεια εκτελούνται οι αλγόριθμοι **UCS** και **A*** μέχρι την εύρεση μίας αποδεκτής τελικής κατάστασης (σύμφωνης με τους κανόνες του προβλήματος) και τυπώνονται οι ενδιάμεσες καταστάσεις της λύσης που βρέθηκε για τους δύο αλγόριθμους. Τέλος για κάθε αλγόριθμο εκτυπώνεται ο αριθμός των κόμβων που χρειάστηκε να επισκεφθεί καθώς και ο χρόνος εκτέλεσης που απαιτήθηκε.

Οι υλοποιήσεις των δύο αλγορίθμων χρησιμοποιούν τη συνάρτηση γενικής αναζήτησης που περιγράφηκε παραπάνω με τις απαραίτητες τροποποιήσεις παραμέτρων για κάθε αλγόριθμο. Η πληροφορία για κάθε κόμβο αποθηκεύεται σε ένα αντικείμενο τύπου **Node**, το οποίο περιέχει τη συμβολοσειρά που παριστάνει την τωρινή κατάσταση, τη γονική κατάσταση, το τρέχον κόστος υπολογισμένο από την αρχική κατάσταση καθώς και την τιμή της ευρετικής συνάρτησης από τον τρέχοντα κόμβο έως την τελική κατάσταση. Στην περίπτωση του αλγόριθμου **UCS** η τιμή της ευρετικής συνάρτησης παραμένει πάντα στο 0. Η αναζήτηση συνεχίζεται δημιουργώντας τα παιδιά της τρέχουσας κατάστασης υπολογίζοντας το κόστος τους, και στην περίπτωση του **A*** υπολογίζεται επιπρόσθετα και η τιμή της ευρετικής συνάρτησης.

Ο αλγόριθμος **UCS** εξ' ορισμού βρίσκει τη βέλτιστη λύση με το μικρότερο κόστος διαδρομής. Το ίδιο συμβαίνει και για τον αλγόριθμο **A*** ο οποίος βρίσκει διαδρομή ίδιου κόστους με τον αλγόριθμο **UCS** από τη στιγμή που η ευρετική συνάρτηση που χρησιμοποιήθηκε είναι αποδεκτή. Όμως αυτή δεν είναι απαραίτητα η ίδια. Ο αλγόριθμος **A*** με την τελική επιλογή ευρετικής συνάρτησης αναπτύσσει αισθητά λιγότερους κόμβους με αποτέλεσμα να έχει καλύτερους χρόνους εκτέλεσης.

Άσκηση 2

Στη δεύτερη άσκηση υλοποιήθηκε ο αλγόριθμος **MINIMAX** στα πλαίσια του παιγνίου δύο παικτών SOS. Το παίγνιο αυτό αποτελεί ακολουθιακό παίγνιο μηδενικού αθροίσματος και τέλειας πληροφορίας. Για το λόγο αυτό η χρήση του αλγορίθμου **MINIMAX** με σκοπό την υλοποίηση ενός παίκτη υπολογιστή είναι εφικτή. Σκοπός του παίγνιου, στη συγκεκριμένη ζητούμενη παραλλαγή, είναι σε ένα πλέγμα διαστάσεων 3x3 να δημιουργηθεί η ακολουθία των συμβόλων SOS είτε οριζόντια, είτε κάθετα, είτε διαγώνια. Ο παίκτης που θα δημιουργήσει την παραπάνω ακολουθία κηρύσσεται νικητής.

Αλγόριθμος

Ο αλγόριθμος χρησιμοποιεί ένα δέντρο αποφάσεων, όπου αποθηκεύει όλες τις πιθανές καταστάσεις του παιγνίου, και έπειτα με έναν αλγόριθμο προσπέλασης κόμβων δέντρων όπως ο **BFS** ή ο **DFS** αναζητεί τη βέλτιστη γι αυτόν επόμενη επιλογή κίνησης, βάσει μίας ευρετικής συνάρτησης αξίας, και επιστρέφει την επιλογή αυτή. Αξίζει να αναφερθέι πως η επιλογή της ευρετικής αυτής συνάρτησης, είναι σημαντική καθώς αποτελεί τον τρόπο αξιολόγησης των τελικών καταστάσεων του παιγνίου και επηρεάζει άμεσα, όχι μόνο την επιλογή της επόμενης κίνησης του παίκτη, αλλά και το πόσο βαθιά θα είναι η αναζήτηση στο δέντρο αποφάσεων.

Η επιλογή της ευρετικής συνάρτησης αξίας έγινε με βάση τον κανόνα τερματισμού του παιγνίου, λαμβάνοντας υπόψη επιπρόσθετα και το βάθος της απόφασης στο δέντρο απόφασης του αλγόριθμου.

Πιο συγκεκριμένα ορίστηκαν οι εξής τιμές για την αξία των τελικών καταστάσεων:

- *(10 – **depth**)*, αν ο παίκτης υπολογιστής εκτελεί νικητήρια κίνηση σε βάθος απόφασης ***depth***.
- *(-10 + **depth**)*, αν ο παίκτης χρήστης εκτελεί νικητήρια κίνηση σε βάθος απόφασης ***depth***.
- *0*, αν το παίγνιο λήγει σε ισοπαλία (κανένας παίκτης δεν έχει νικήσει και το πλέγμα είναι γεμάτο).

Κάνοντας αυτή την αξιολόγηση, η ευρετική συνάρτηση αξίας μπορεί να δώσει στις τερματικές καταστάσεις ως τιμές τους ακέραιους στο διάστημα [-9, -2] για νικητήρια κίνηση του παίκτη χρήστη, 0 για ισοπαλία και τους ακέραιους στο διάστημα [2, 9] για νικητήρια κίνηση του παίκτη υπολογιστή. Εδώ χρησιμοποιήθηκε το γεγονός πως το μέγιστο βάθος του δέντρου απόφασης είναι 8 (στην αρχή του παιχνιδιού υπάρχουν 8 διαθέσιμες θέσεις για τοποθέτηση είτε S είτε O).

Με αυτόν τον τρόπο δίνεται προτεραιότητα στην επιλογή κινήσεων που οδηγούν

τον παίκτη υπολογιστή σε νίκες που βρίσκονται σε μικρό βάθος απόφασης, "τιμωρώντας" παράλληλα την επιλογή κινήσεων που οδηγούν τον παίκτη υπολογιστή σε ήττες που βρίσκονται σε μικρό βάθος απόφασης.

Υλοποίηση

Η υλοποίηση του αλγορίθμου έγινε σε γλώσσα **Java** και έγινε με τη χρήση αναδρομής, όπως αναφέρεται και στην εκφώνηση.

Με την εκτέλεση του προγράμματος ο χρήστης έχει την επιλογή να πληκτρολογήσει τις εξής τρεις εντολές:

Η πρώτη εντολή είναι η **start name_of_p1 name_of_p2**, η οποία ξεκινά ένα παίγνιο **SOS** μεταξύ των παικτών **p1** και **p2** των οποίων τα ονόματα επιλέγει ο χρήστης, με πρώτο παίκτη να αποτελεί ο παίκτης **p1**. Άν η επιλογή του ονόματος είναι **minimax** τότε ο παίκτης είναι ο υπολογιστής και η επιλογή κινήσεών του γίνεται με βάση τον αλγόριθμο **MINIMAX**. Άν η επιλογή του ονόματος είναι **cpu** τότε ο παίκτης είναι επίσης ο υπολογιστής και η επιλογή κινήσεών του γίνεται επιλέγοντας τυχαία τις συντεταγμένες και το σύμβολο σε κάθε γύρο. Σε οποιαδήποτε άλλη επιλογή ονόματος, ο παίκτης είναι ο χρήστης. Αν και στην άσκηση ζητείται η υλοποίηση της περίπτωσης που ο παίκτης **MAX** (παίκτης υπολογιστής) παίζει πρώτος, η υλοποίηση του αλγόριθμου επιτρέπει στον παίκτη υπολογιστή να παίξει επίσης δεύτερος. Με αυτόν τον τρόπο, είναι εφικτός οποιοσδήποτε συνδυασμός παικτών για το παίγνιο. Για παράδειγμα, πληκτρολογώντας την εντολή **start minimax minimax**, δύο παίκτες-υπολογιστές με επιλογή κινήσεων με τον αλγόριθμο **MINIMAX** παίζουν μεταξύ τους.

Η δεύτερη εντολή είναι η **help**, η οποία εμφανίζει στην οθόνη τους κανόνες του παίγνιου, καθώς και τις αποδεκτές εντολές που μπορεί να πληκτρολογήσει ο χρήστης.

Η τρίτη εντολή είναι η **exit**, με την οποία τερματίζεται το πρόγραμμα.

Κατα την εκχώρηση των εντολών γίνεται έλεγχος για την ορθότητά τους, και άν δε συμβαδίζουν με τις ήδη υπάρχουσες εντολές ξαναζητείται η εκχώρησή τους. Για παράδειγμα δεν επιτρέπεται η εκχώρηση των ονομάτων **start**, **help** και **exit** ως ονόματα για τους παίκτες, όταν εκχωρείται η εντολή **start**.

Αρχικά, υλοποιήθηκε το παιχνίδι μεταξύ δύο χρηστών, και αφότου βεβαιώθηκε η σωστή λειτουργία του, υλοποιήθηκε ο αλγόριθμος **MINIMAX**. Η πληροφορία αποθηκεύεται σε αντικείμενα τύπου **Node**, τα οποία έχουν ως πεδία ένα διδιάστατο πίνακα από **char**, όπου αποθηκεύεται η τρέχουσα κατάσταση του παιγνίου, και ένα **ArrayList** από **Node** αντικείμενα στο οποίο αποθηκεύονται όλες ο επόμενες καταστάσεις παιδιά του δέντρου απόφασης. Επίσης η κλάση **Node** έχει μια μέθοδο **findChildern()** η οποία βρίσκει όλες τις επόμενες πιθανές καταστάσεις του παγνίου. Η εξερεύνηση του δέντρου γίνεται με τη χρήση μιας παραλλαγής του αλγόριθ-

μου **DFS** και η δημιουργία των κόμβων του δέντρου απόφασης γίνεται δυναμικά με σκοπό την αποφυγή της υπερχείλισης της μνήμης του **JVM** λόγω του μεγάλου όγκου των αντικειμένων που δημιουργούνται κάθε φορά, χωρίς να χρειαστεί να αυξήσουμε το χώρο του **heap/stack** κατά την εκτέλεση του προγράμματος.

Όταν ένα παίγνιο βρίσκεται σε εξέλιξη, εάν ο τρέχον παίκτης είναι παίκτης-υπολογιστής, εκτελείται η μέθοδος **stopFlow()**, η οποία σταματά τη ροή του παίγνιου μέχρι ο χρήστης να πατήσει **Enter**, έτσι ώστε να έχει τη δυνατότητα να δεί το πλέγμα. Άν ο τρέχον παίκτης, είναι ο χρήστης, πρέπει να πληκτρολογήσει τις συντεταγμένες όπου επιθυμεί να τοποθετήσει, είτε το σύμβολο **S** είτε το σύμβολο **O**, καθώς και το σύμβολο που επιθυμεί, χωρισμένα με κενό (π.χ 2 2 O).

Και εδώ, άν δε δοθεί η συμβολοσειρά των συντεταγμένων και του συμβόλου με τη μορφοποίηση που αναφέρθηκε παραπάνω, ξαναζητείται η εκχώρησή της από το χρήστη.

Τέλος, σε κάθε κίνηση του παίκτη-υπολογιστή με όνομα **minimax**, εκτυπώνονται ο αριθμός των κόμβων που εξερεύνησε ο αλγόριθμος **MINIMAX** καθώς και ο χρόνος που απαιτήθηκε για να παρθεί η απόφαση, σε μορφοποίηση ss.SSS (seconds.milliseconds).

Για την επιβεβαίωση της σωστής λειτουργίας της υλοποίησης του αλγόριθμου, μεταξύ άλλων συγκρίθηκε ο αριθμός όλων των πιθανών καταστάσεων του παιγνίου στο πλήρες δέντρο καταστάσεων, με τον πραγματικό αριθμό καταστάσεων που επισκέπτεται ο αλγόριθμος κατά την υλοποίησή του στο δέντρο απόφασης. Πιο συγκεκριμένα, υπολογίστηκε ο αριθμός των κόμβων του πλήρους δένδρου καταστάσεων (πλην της ρίζας) σε κάποια διαμόρφωση του πλέγματος, ο οποίος προκύπτει από το άθροισμα:

$$\sum_{i=1}^{\alpha} \left[\left(\prod_{n=\alpha+1-i}^{\alpha} n \right) 2^i \right]$$

όπου το α είναι ο αριθμός των κενών κελιών σε κάποια διαμόρφωση του πλέγματος.

Για παράδειγμα, όταν επιλέγεται η πρώτη κίνηση (όπου υπάρχουν 8 κενά κελιά στη συγκεκριμένη διαμόρφωση), το πλήρες δέντρο καταστάσεων περιλαμβάνει, με βάση τον παραπάνω τύπο, 17.017.968 κόμβους (πλην της ρίζας). Η υλοποίηση του αλγόριθμου επισκέπτεται 10.449.792 κόμβους (πλην της ρίζας), κάτι το οποίο είναι αναμενόμενο, αφού υπάρχουν και καταστάσεις σε βάθος μικρότερο από 8, όπου το παίγνιο λήγει με νίκη κάποιου παίκτη ο οποίος έχει σχηματίσει αλυσίδα **SOS**, χωρίς να έχει συμπληρωθεί πλήρως το πλέγμα του παίγνιου. Έτσι τελικά, ο αριθμός των κόμβων που ελέγχει ο αλγόριθμος στο δέντρο απόφασης, είναι πάντα μικρότερος, ή ίσος από τον αριθμό των κόμβων του πλήρους δέντρου καταστάσεων.

Γίνεται άμεσα αντιληπτό πως επειδή ο όγκος των διαμορφώσεων που καλείται να ελέγξει ο παίκτης υπολογιστής είναι πολύ μεγάλος στις αρχικές κινήσεις, ο χρόνος που απαιτείται για την επεξεργασία τους είναι σαφώς μεγαλύτερος από το

χρόνο που χρειάζεται σε μετέπειτα κινήσεις. Δίνεται παρακάτω παράδειγμα ενός ενδεικτικού παιχνιδιού σε γύρους όπου ο υπολογιστής παίζει πρώτος με τον αριθμό διαμορφώσεων που ελέγθηκαν σε κάθε γύρο.

0. Αρχική κατάσταση: 2 1 O

1. Κίνηση παίκτη **minimax**: 1 1 O
Διαμορφώσεις που ελέγθηκαν: 10.449.792

2. Κίνηση παίκτη χρήστη: 2 2 S

3. Κίνηση παίκτη **minimax**: 1 2 S
Διαμορφώσεις που ελέγθηκαν: 69.228

4. Κίνηση παίκτη χρήστη: 3 2 S

5. Κίνηση παίκτη **minimax**: 1 3 S
Διαμορφώσεις που ελέγθηκαν: 620

6. Κίνηση παίκτη χρήστη: 3 3 O

7. Κίνηση παίκτη **minimax**: 2 3 S
Διαμορφώσεις που ελέγθηκαν: 12

8. Κίνηση παίκτη χρήστη: 3 1 O
Λήξη του παίγνιου σε ισοπαλία