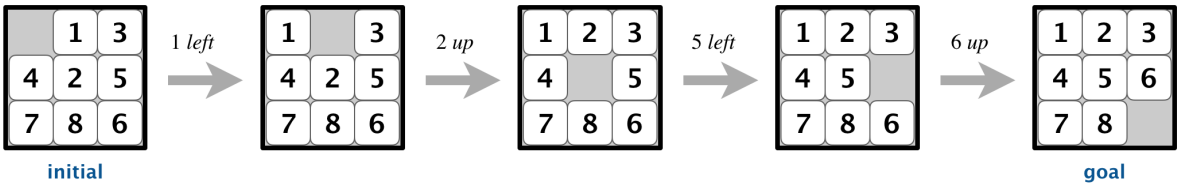
Πρώτη Προαιρετική Εργασία - Τεχνητή Νοημοσύνη

Βασίλειος Παπαστέργιος (ΑΕΜ: 3651)

4 Απριλίου, 2021



Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης
Τμήμα Πληροφορικής

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάθημα:	Τεχνητή Νοημοσύνη
Εξάμηνο:	4ο
Καθηγητές:	Ιωάννης Βλαχάβας
	Δημήτριος Βράκας

Contents

1 Εισαγωγή

Το παρόν έγγραφο αποτελεί γραπτή αναφορά για την πρώτη προαιρετική εργασία του μαθήματος της Τεχνητής Νοημοσύνης. Παραδίδεται συμπληρωματικά με τα αρχεία πηγαίου κώδικα σε γλώσσα προγραμματισμού C++, τα οποία αναπτύχθηκαν από τον γράφοντα ως προγραμματιστική λύση της εργασίας. Στις σελίδες που ακολουθούν, παρατίθεται τεκμηρίωση επί της θεωρητικής μοντελοποίησης του N-puzzle προβλήματος, καθώς και περαιτέρω ανάλυση επί της υλοποίησης των αλγορίθμων αναζήτησης σε γλώσσα προγραμματισμού C++.

2 Περιγραφή του προβλήματος

Στην Τεχνητή Νοημοσύνη, το πρωταρχικό στάδιο για την επίλυση ενός προβλήματος έγκειται στην κατάλληλη και επαρκή περιγραφή του ίδιου του προβλήματος. Για αυτόν ακριβώς το λόγο, σε αυτή την ενότητα θα προσπαθήσουμε να οριοθετήσουμε τον κόσμο του N-puzzle προβλήματος και να περιγράψουμε τα συγκεκριμένα χαρακτηριστικά του.

2.1 Ο κόσμος του N-puzzle προβλήματος

Το πρόβλημα του N-puzzle αποτελεί ένα από τα πλέον διαδεδομένα *toy problems* στον τομέα της Τεχνητής Νοημοσύνης. Σύμφωνα με τους κανόνες, το παιχνίδι διαδραματίζεται επάνω σε ένα πλαίσιο με πλακίδια. Η αρίθμηση των πλακιδίων ξεκινά από το 1, ενώ το πλαίσιο αποτελείται από `WIDTH` στήλες και `HEIGHT` γραμμές συνολικά.

Το πλαίσιο, βέβαια, δεν είναι πλήρως γεμάτο με πλακίδια. Για την ακρίβεια, μια θέση πλακιδίου είναι κενή, επιτρέποντας στον παίκτη να σύρει κάποιο από τα γειτονικά πλακίδια, αλλάζοντας τη διάταξη των πλακιδίων επάνω στο πλαίσιο. Με άλλα λόγια, τα πλακίδια μπορούν να γλιστρήσουν ανάμεσα στα άλλα, εφόσον μια θέση είναι κενή. Αξιοποιώντας, λοιπόν, αυτή τη δυνατότητα αναδιάταξης, ο παίκτης καλείται, ξεκινώντας από μια αρχική διάταξη των πλακιδίων, να φτάσει στη λύση του puzzle.

Στο πλαίσιο της ανάλυσής μας για την παρούσα εργασία θα θεωρήσουμε πως η *λύση* του puzzle είναι η κατάλληλη διάταξη των πλακιδίων, τέτοια ώστε οι αριθμοί να εμφανίζονται σε αύξουσα σειρά (ξεκινώντας από την επάνω αριστερή γωνία) και το κενό να βρίσκεται στο τέλος του πλαισίου (κάτω δεξιά γωνία). Για παράδειγμα, στο ακόλουθο σχεδιάγραμμα, το στιγμιότυπο του πλαισίου στα αριστερά μπορεί να θεωρηθεί ως μια (τυχούσα) αρχική κατάσταση του προβλήματος, ενώ το αντίστοιχο στα δεξιά ως η τελική κατάσταση ή κατάσταση-στόχος, που σηματοδοτεί και τη λύση του puzzle.

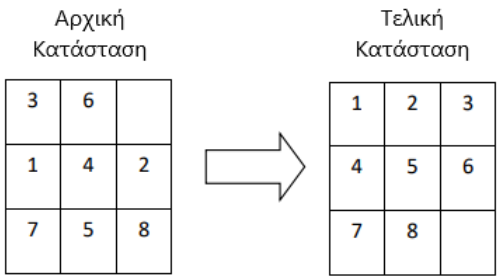


Figure 1: Παράδειγμα αρχικής και τελικής κατάστασης ενός 8-puzzle προβλήματος

Όπως γίνεται φανερό, μια κατάσταση του N-puzzle προβλήματος μπορεί εύκολα να περιγραφεί ως το *στιγμιότυπο* της διάταξης των πλακιδίων επάνω στο πλαίσιο, μια δεδομένη χρονική στιγμή. Εναλλακτικά, μια κατάσταση του προβλήματος αντιστοιχίζεται με μια *φωτογραφία* του πλαισίου, που απαθανατίζει τη θέση των αριθμών επάνω στο πλαίσιο, μια συγκεκριμένη στιγμή της διαδικασίας επίλυσης.

Έχοντας ορίσει την θεωρητική αναπαράσταση των καταστάσεων του προβλήματος, ο *τρόπος μετάβασης* μεταξύ αυτών ακολουθεί ως λογικό επόμενο της σκέψης μας. Το σημείο αυτό είναι το πλέον κατάλληλο, προκειμένου να εισάγουμε στη συλλογιστική μας τους *τελεστές μετάβασης* του προβλήματος, δηλαδή τις διαθέσιμες ενέργειες που μπορεί να εκτελέσει ο παίκτης, για να αναδιατάξει τα πλακίδια.

Κατά τη διάρκεια της διαδικασίας αυτής, θα εστιάσουμε αποκλειστικά στο κενό του πλαισίου, και πιο συγκεκριμένα στις διαθέσιμες μεταβάσεις του. Για λόγους ομοιομορφίας και μόνο, θα θεωρήσουμε πως το κενό που υπάρχει στο πλαίσιο είναι και αυτό ένα (νοητό) πλακίδιο. Επομένως, κατά σύμβαση, θα αναφερόμαστε εφεξής καταχρηστικά σε αυτό με τον όρο «*κενό πλακίδιο*», χωρίς αυτό να επηρεάζει την υπόλοιπη θεωρητική σκιαγράφηση του προβλήματος.

Από τον ορισμό του προβλήματος στο φυσικό κόσμο, λαμβάνουμε ως δεδομένο πως οι διαθέσιμες κινήσεις του κενού πλακιδίου είναι οι εξής τέσσερις:

- μετακίνηση προς τα **επάνω**
- μετακίνηση προς τα **κάτω**
- μετακίνηση προς τα **αριστερά**
- μετακίνηση προς τα **δεξιά**

Και οι τέσσερις αυτές κινήσεις συνεπάγονται και αντίθετη κίνηση ενός άμεσα εμπλεκόμενου (μη κενού) πλακιδίου. Για παράδειγμα, η κίνηση του κενού πλακιδίου προς τα επάνω, συνεπάγεται την κίνηση προς τα κάτω του πλακιδίου που βρισκόταν ακριβώς πάνω από το κενό, καλύπτοντας τη θέση του. Εντελώς αντίστοιχα, η κίνηση του κενού πλακιδίου προς τα δεξιά, περιλαμβάνει μετακίνηση του δεξιού του πλακιδίου προς τα αριστερά. Αντίστοιχες ανταλλαγές πλακιδίων ισχύουν και για τις υπόλοιπες κινήσεις.

Λαμβάνοντας αυτά υπ' όψιν, γίνεται φανερό πως υπάρχουν περιορισμοί για την εκτέλεση των παραπάνω ενεργειών. Πιο συγκεκριμένα, το κενό πλακίδιο δεν μπορεί να μετακινηθεί προς τα επάνω, όταν βρίσκεται ήδη στην πρώτη γραμμή του πλαισίου. Παρομοίως, δεν μπορεί να μετακινηθεί προς τα κάτω, όταν βρίσκεται ήδη στην τελευταία γραμμή, ενώ αντίστοιχες διαπιστώσεις ισχύουν για τις κινήσεις προς τα αριστερά & προς τα δεξιά, με περιορισμό ως προς την ευρισκόμενη στήλη. Έτσι, οι διαθέσιμες κινήσεις του κενού πλακιδίου μπορούν να εμπλουτιστούν ως εξής:

- μετακίνηση προς τα **επάνω**, *όταν δεν βρίσκεται στην πρώτη γραμμή*
- μετακίνηση προς τα **κάτω**, *όταν δεν βρίσκεται στην τελευταία γραμμή*
- μετακίνηση προς τα **αριστερά**, *όταν δεν βρίσκεται στην πρώτη στήλη*
- μετακίνηση προς τα **δεξιά**, *όταν δεν βρίσκεται στην τελευταία στήλη*

Συνολικά, ζητούμενο της προσέγγισής μας είναι να αξιοποιήσουμε αυτή την θεωρητική μοντελοποίηση του προβλήματος, καταλήγοντας σε ένα σύστημα Τεχνητής Νοημοσύνης που μπορεί να επιλύει ένα τυχόν, δοθέν N-puzzle. Ειδικότερα, αναζητείται η ακριβής σειρά ενεργειών που απαιτούνται από τον παίκτη, προκειμένου να οδηγηθεί από την δοθείσα αρχική κατάσταση στην κατάσταση-στόχο. Η διαπίστωση αυτή κατατάσσει το N-puzzle πρόβλημα στην ευρύτερη κατηγορία των *προβλημάτων σχεδιασμού ενεργειών*.

2.2 Εφαρμογή της μοντελοποίησης σε C++

Αξιοποιώντας όσα αναλύθηκαν σε θεωρητικό επίπεδο, σε αυτή την ενότητα θα προσπαθήσουμε να παρουσιάσουμε την υλοποίηση αυτών σε γλώσσα προγραμματισμού C++. Τα παραδοτέα αρχεία κώδικα που αναλύονται στην παρούσα ενότητα είναι τα αρχεία `State.h` και `State.cpp`.

Ειδικότερα, στην ενότητα αυτή εστιάζουμε στην υλοποίηση της γενικής οντότητας «*κατάσταση*» του N-puzzle προβλήματος. Ήδη από τον ορισμό του προβλήματος στον φυσικό κόσμο, γίνεται φανερό πως αυτή θα πρέπει να περιλαμβάνει ένα στιγμιότυπο της τρέχουσας διάταξης των πλακιδίων επάνω στο πλαίσιο. Σε επίπεδο υλοποίησης, το στιγμιότυπο αυτό πρόκειται για έναν δισδιάστατο πίνακα μη προσημασμένων, ακεραίων αριθμών. Ο πίνακας φέρει το χαρακτηριστικό όνομα ***board***.

Από προεπιλογή, ο πίνακας αυτός αρχικοποιείται σε ένα προκαθορισμένο 8-puzzle, το οποίο είναι επιλύσιμο. Η αρχικοποίηση αυτή είναι ενδεικτική, ενώ θα μπορούσε, χωρίς πρόβλημα, να είναι ένα οποιοδήποτε στιγμιότυπο N-puzzle προβλήματος. Σε κάθε περίπτωση, ο πίνακας αυτός αποθηκεύει τη «φωτογραφία» του πλαισίου, την δεδομένη χρονική στιγμή.

Συμπληρωματικά με τον δισδιάστατο πίνακα, η κλάση αποθηκεύει δύο επιπλέον θετικούς ακεραίους αριθμούς, οι οποίοι αναπαριστούν τη γραμμή και τη στήλη, αντίστοιχα, στην οποία βρίσκεται το κενό πλακίδιο, τη δεδομένη

χρονική στιγμή. Ο λόγος για τις μεταβλητές ***blankTileRow*** και ***blankTileColumn***. Για λόγους πληρότητας, κρίνεται σκόπιμο να επισημανθεί πως η αρίθμηση των γραμμών και των στηλών ξεκινά από τον αριθμό 0 (0-indexed), ενώ το σημείο $(0, 0)$ αντιστοιχεί στην επάνω αριστερή γωνία του πλαισίου.

Ένα επιπλέον χαρακτηριστικό της κατάστασης είναι το βάθος (***depth***). Πρακτικά, το βάθος αντιπροσωπεύει το πλήθος των κινήσεων που απαιτούνται από την αρχική κατάσταση, προκειμένου να οδηγηθούμε στην τρέχουσα. Έτσι, η αρχική κατάσταση έχει βάθος ίσο με το 0, οι καταστάσεις που παράγονται άμεσα (σε ένα βήμα) από την αρχική έχουν βάθος 1 κ.ο.κ.

Τέλος, κάθε κατάσταση, πλην της αρχικής, διαθέτει έναν δείκτη προς τη γονική της κατάσταση (***previous***). Με τον όρο «γονική» εννοούμε την κατάσταση από την οποία προήλθε η τρέχουσα, μετά την εκτέλεση μόνο μίας από τις έγκυρες, διαθέσιμες ενέργειες.

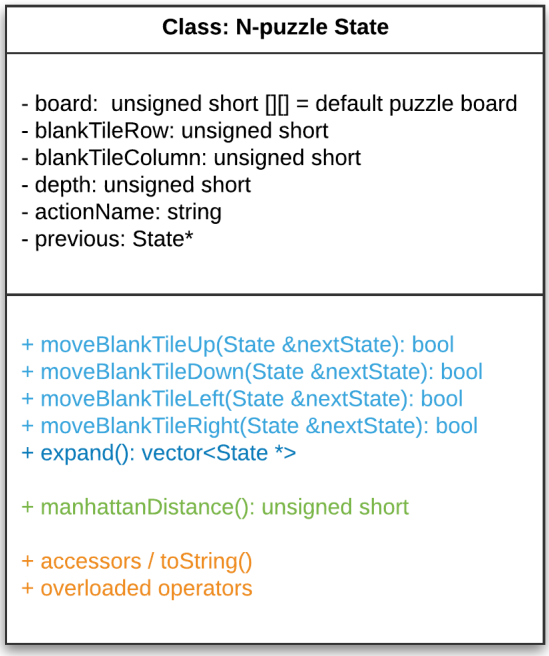


Figure 2: UML διάγραμμα για την κλάση που αναπαριστά μια κατάσταση του προβλήματος

Μια ενδιαφέρουσα παρατήρηση αφορά τον τρόπο υπολογισμού του βάθους μιας δεδομένης κατάστασης. Η προσέγγιση που υιοθετήθηκε στην παρούσα μοντελοποίηση είναι ο ταυτόχρονος (on the fly) υπολογισμός του βάθους και η αποθήκευσή του σε ένα πεδίο της κλάσης. Η εναλλακτική προσέγγιση θα ήταν ο υπολογισμός να πραγματοποιείται με επαναληπτικό τρόπο (μέσω της καταμέτρησης των γονικών κόμβων έως την αρχική), χωρίς αυτό να αποθηκεύεται σε κάποιο πεδίο της κλάσης. Στη δεύτερη περίπτωση, το βάθος μιας κατάστασης υπολογίζεται «από την αρχή», κάθε φορά που αυτό χρειάζεται, ακόμη και αν έχει ήδη υπολογιστεί παλιότερα για τη συγκεκριμένη κατάσταση.

Στο πλαίσιο της παρούσης εργασίας, προτιμήθηκε η υλοποίηση της πρώτης τακτικής. Ο λόγος είναι πως δόθηκε προτεραιότητα στην ταχύτητα εκτέλεσης του συστήματος και, κατ΄ επέκταση, στην ταχύτητα εύρεσης της λύσης του δοθέντος (puzzle). Είναι εμφανές πως η τακτική αυτή έχει αντίκτυπο σε χρήση μνήμης από το σύστημά μας. Ο αντίκτυπος είναι μεγαλύτερος όσο μεγαλώνει ο αριθμός N του puzzle, ενώ είναι και άμεσα εξαρτώμενος από τον αλγόριθμο αναζήτησης που εφαρμόζεται. Οι αλγόριθμοι αναζήτησης θα αναλυθούν εκτενώς στο επόμενο κεφάλαιο.

Περνώντας σε επίπεδο *δημοσίων μεθόδων*, η κλάση υποστηρίζει τρεις κατηγορίες τέτοιων, οι οποίες έχουν ομαδοποιηθεί χρωματικά, όπως φαίνεται και στο παραπάνω διάγραμμα. Οι κατηγορίες μεθόδων είναι οι εξής:

- μέθοδοι εφαρμογής ενέργειας (μπλε)
- μέθοδος υλοποίησης ευριστικής συνάρτησης (πράσινο)

- βοηθητικές μέθοδοι και υπερφορτωμένοι τελεστές (πορτοκαλί)

Η πλέον σημαντική κατηγορία είναι οι μέθοδοι εφαρμογής ενέργειας. Με πλήρη πιστότητα στο θεωρητικό μοντέλο που κατασκευάσαμε, οι μέθοδοι αυτές αναπαριστούν τις τέσσερις διαθέσιμες κινήσεις που μπορεί να εκτελέσει το κενό πλακίδιο. Στο εσωτερικό των μεθόδων αυτών περικλείεται και ο έλεγχος εγκυρότητας της εκάστοτε κίνησης. Αυτός είναι και ο λόγος που ο επιστρεφόμενος τύπος των μεθόδων είναι `bool`, υποδηλώνοντας αν η ενέργεια είναι έγκυρη και μπορεί να εφαρμοστεί ή όχι.

Η μπλε κατηγορία περιλαμβάνει, επίσης, μια μέθοδο που ενεργεί συνολικά επί των τεσσάρων προαναφερθέντων. Πρόκειται για τη μέθοδο `expand`, η οποία ελέγχει όλες τις διαθέσιμες ενέργειες που μπορούν να εκτελεστούν στην τρέχουσα κατάσταση και επιστρέφει μια λίστα από καταστάσεις. Επί της ουσίας, είναι οι καταστάσεις εκείνες που προκύπτουν από την εφαρμογή όλων των **έγκυρων** ενεργειών επί της τρέχουσας. Με άλλα λόγια, επεκτείνει την τρέχουσα κατάσταση, κατά όσες κατευθύνσεις επιτρέπουν οι περιορισμοί που διέπουν τις ενέργειες.

Αναφορικά με τις υπόλοιπες μεθόδους, η γενική εικόνα είναι πως δεν επιδρούν άμεσα στην μοντελοποίηση του προβλήματος. Η μεν υλοποίηση της ευριστικής συνάρτησης βρίσκει άμεση εφαρμογή σε ορισμένους από τους αλγορίθμους αναζήτησης που θα συζητήσουμε στο αμέσως επόμενο κεφάλαιο. Η δε τελευταία κατηγορία έχει επικουρικό ρόλο και συνεισφέρει στην αναγνωσιμότητα του κώδικα, χωρίς ιδιαίτερη πρακτική σημασία στη διαδικασία της μοντελοποίησης.

3 Αλγόριθμοι αναζήτησης

Έχοντας αναλύσει ενδελεχώς τη μοντελοποίηση του προβλήματος, η διαδικασία αναζήτησης της λύσης με συστηματικό τρόπο είναι εκείνη που ακολουθεί νοηματικά. Στο κεφάλαιο αυτό, θα αναφερθούμε στους αλγορίθμους αναζήτησης που χρησιμοποιήθηκαν, προσπαθώντας να εμβαθύνουμε στην υλοποίησή τους σε γλώσσα προγραμματισμού C++, καθώς και να συγκρίνουμε την απόδοσή τους υπό τρεις διαφορετικές οπτικές. Τα παραδοτέα αρχεία κώδικα που αναλύονται στην παρούσα ενότητα είναι όλα τα αρχεία πηγαίου κώδικα που περιέχονται στο φάκελο `SearchAlgorithms`.

3.1 Γενικά

Οι αλγόριθμοι αναζήτησης που χρησιμοποιήθηκαν στην παρούσα εργασία προέρχονται από δύο διαφορετικές γενικές κατηγορίες. Πιο συγκεκριμένα, επιλέχθηκαν και χρησιμοποιήθηκαν οι εξής αλγόριθμοι:

- Αναζήτηση πρώτα κατά βάθος (DFS) και αναζήτηση πρώτα κατά πλάτος (BFS), από την κατηγορία των **τυφλών αλγορίθμων**
- Αναζήτηση πρώτα προς το καλύτερο (BestFS) και άλφα-άστρο (A*), από την κατηγορία των **αλγορίθμων ευρετικής αναζήτησης**

Στο πλαίσιο της θεώρησής μας, οι παραπάνω αλγόριθμοι αναζήτησης προσεγγίστηκαν υπό ένα αφαιρετικό, αντικειμενοστραφές πρίσμα. Στις υποενότητες που ακολουθούν, θα προσπαθήσουμε να τεκμηριώσουμε τον χαρακτηρισμό αυτό.

3.2 Το αφαιρετικό πρότυπο αναζήτησης λύσης

Η Πληροφορική χαρακτηρίζεται δικαίως ως η επιστήμη των αφαιρέσεων. Στο πλαίσιο αυτό, λοιπόν, οι τέσσερις αλγόριθμοι αναζήτησης που αναπτύχθηκαν στην παρούσα εργασία αντιμετωπίστηκαν νοηματικά ως υποκατηγορίες ενός γενικότερου προτύπου συστηματικής αναζήτησης της λύσης. Το πρότυπο αυτό θα μπορούσε να χαρακτηριστεί ως μια αφαιρετική αναπαράσταση των κοινών στοιχείων που μοιράζονται οι τέσσερις αλγόριθμοι. Οι τελευταίοι, βασιζόμενοι σε αυτό το πλαίσιο ως κορμό, προσφέρουν εξειδικεύσεις σε ορισμένα χαρακτηριστικά, γεγονός που διακρίνει τελικά τους αλγορίθμους μεταξύ τους.

Πιο συγκεκριμένα, σε όλους τους αλγορίθμους αναζήτησης που παρουσιάστηκαν, διακρίνονται δύο διαφορετικές δομές δεδομένων, οι οποίες εξυπηρετούν τη διαδικασία της αναζήτησης. Η πρώτη εξ αυτών, αναπαριστά το *μέτωπο αναζήτησης* (search frontier). Με τον όρο αυτό εννοούμε το σύνολο των καταστάσεων του προβλήματος που ο αλγόριθμος έχει ανακαλύψει την τρέχουσα χρονική στιγμή και πρόκειται να ερευνηθούν στο άμεσο μέλλον. Η έρευνα αυτή λαμβάνει τη μορφή ελέγχου για το αν η κατάσταση είναι τελική ή όχι. Στη δεύτερη περίπτωση, η (μη τελική) κατάσταση επεκτείνεται (expand) και προκύπτουν παράγωγες καταστάσεις, με βάση τις έγκυρες, διαθέσιμες ενέργειες που έχουν οριστεί στο θεωρητικό μας μοντέλο.

Η δεύτερη δομή που αποτελεί κοινό χαρακτηριστικό μεταξύ των τεσσάρων αλγορίθμων αναζήτησης είναι το επονομαζόμενο *κλειστό σύνολο* (closed set ή already expanded set). Όπως υποδηλώνει και το όνομά του, το σύνολο αυτό περιλαμβάνει τις καταστάσεις που δεν είναι τελικές και έχουν ήδη ερευνηθεί και επεκταθεί την τρέχουσα χρονική στιγμή από τον αλγόριθμο. Το κλειστό σύνολο διαδραματίζει κομβικό ρόλο στη διασφάλιση του κριτηρίου της περατότητας των αλγορίθμων αναζήτησης.

Λαμβάνοντας υπόψιν τα δύο αυτά σύνολα, μπορούμε εύκολα να ορίσουμε το γενικό κορμό δράσης όλων αλγορίθμων αναζήτησης. Έτσι, στο μέτωπο αναζήτησης τοποθετείται, αρχικά, η δοθείσα αρχική κατάσταση, ενώ το κλειστό σύνολο, αρχικά, είναι κενό.

Κατόπιν, για όσο χρόνο το μέτωπο αναζήτησης περιλαμβάνει καταστάσεις που δεν έχουν ερευνηθεί, ο αλγόριθμος αποσπά την πρώτη κατάσταση από το μέτωπο και την εξετάζει. Σε περίπτωση που η κατάσταση είναι ισοδύναμη με την κατάσταση-στόχο, τότε η λύση έχει βρεθεί και ο αλγόριθμος σταματά. Σε αντίθετη περίπτωση, η κατάσταση επεκτείνεται εφαρμόζοντας σε αυτήν τους τελεστές μετάβασης. Αποτέλεσμα αυτής της διαδικασίας είναι η δημιουργία ενός αριθμού παράγωγων καταστάσεων, που ονομάζονται καταστάσεις-παιδιά (children states). Η γονική κατάσταση εισάγεται στο κλειστό σύνολο, ενώ τα παιδιά που δεν ανήκουν στο κλειστό σύνολο (δεν έχουν ερευνηθεί σε προηγούμενο βήμα του αλγορίθμου) εισάγονται στο μέτωπο αναζήτησης.

Γίνεται, επομένως, φανερό ότι ο αλγόριθμος τερματίζει είτε όταν βρεθεί λύση, είτε όταν ερευνηθούν όλες οι πιθανές καταστάσεις, χωρίς να βρεθεί λύση. Η περιγραφόμενη διαδικασία μπορεί να συνοψιστεί στον ακόλουθο ψευδοκώδικα:

```
SearchAlgorithm (initialState, goalState)

    initialize the search frontier data structure;                  // specification #1
    searchFrontier ← initialState;
    alreadyExpanded ← {};  // empty set

    while (search frontier is NOT empty) do
        currentState ← First(searchFrontier);                       // specification #2

        if (currentState is NOT already expanded)
            if (currentState == goalState) then return currentState;

            childrenStates ← expand(currentState);
            add currentState into alreadyExpanded;

            for (childState : childrenStates)
                if (childState is NOT already expanded)
                    evaluate a heuristic function on childState (if needed)
                    add childState into searchFrontier;             // specification #3
    endwhile
    return solution_not_found;
end
```

3.3 Διαφοροποιήσεις επί του αφαιρετικού προτύπου αναζήτησης

Στον ψευδοκώδικα, βέβαια, σημειώνονται τρία σημεία στα οποία η γενική διαδικασία αναζήτησης επιδέχεται εξειδικεύσεων από τους διάφορους τύπους αλγορίθμων. Η πρώτη διαφοροποίηση (specification #1) έγκειται στην επιλογή της δομής δεδομένων που υλοποιεί το μέτωπο αναζήτησης. Πιο συγκεκριμένα, το μέτωπο αναζήτησης υλοποιείται ως:

- στοίβα (stack), στην περίπτωση του DFS
- ουρά (queue), στην περίπτωση του BFS
- ουρά προτεραιότητας/ελαχίστων (priority/min queue), στις περιπτώσεις των BestFS και A*

Σε στενή συσχέτιση με τη δομή δεδομένων που υλοποιεί το μέτωπο αναζήτησης βρίσκεται και η δεύτερη διαφοροποίηση (specification #2). Η διαφοροποίηση αυτή αφορά στον προσδιορισμό του κορυφαίου στοιχείου του μετώπου αναζήτησης, προκειμένου αυτό να εξαχθεί και να ερευνηθεί από τον αλγόριθμο. Με άλλα λόγια, η διαφοροποίηση καθορίζει τη σειρά με την οποία ερευνώνται οι καταστάσεις που ο αλγόριθμος ανακαλύπτει. Λεπτομερέστερα, ως κορυφαίο στοιχείο του μετώπου αναζήτησης θεωρείται:

- το στοιχείο στην κορυφή της στοίβας (stack top element), δηλαδή εκείνο που εισήχθη πιο πρόσφατα στη δομή, στην περίπτωση του DFS
- το μπροστινό στοιχείο της ουράς (queue front element), δηλαδή εκείνο που εισήχθη νωρίτερα από όλα τα υπόλοιπα στη δομή, στην περίπτωση του BFS

- το πρώτο στοιχείο της ουράς ελαχίστων (priority/min queue front element), δηλαδή το ελάχιστο στοιχείο της δομής, στις περιπτώσεις των BestFS και A*

Αντίστοιχα δεδομένα ισχύουν και για την τρίτη διαφοροποίηση (specification #3), η οποία έγκειται στον τρόπο εισαγωγής νέων στοιχείων στη δομή του μετώπου αναζήτησης. Η εισαγωγή αυτή πραγματοποιείται με απόλυτη πιστότητα στους κανόνες που ορίζουν οι τρεις δομές δεδομένων που περιγράφηκαν. Ειδικότερα, όταν ο αλγόριθμος ανακαλύπτει μια νέα κατάσταση που δεν έχει επεκταθεί σε προηγούμενο στάδιο, την εισάγει:

- στην κορυφή της στοίβας, στην περίπτωση του DFS
- στο τέλος της ουράς, στην περίπτωση του BFS
- στο κατάλληλο σημείο της ουράς ελαχίστων, με βάση τη διάταξη, στις περιπτώσεις των BestFS και A*

Ιδιαίτερο ενδιαφέρον στους δύο τελευταίους αλγορίθμους (BestFS και A*) παρουσιάζει η έννοια της διάταξης των καταστάσεων. Πρόκειται για μια έννοια που προϋποθέτει την ύπαρξη ενός τρόπου συστηματικής και ενιαίας αξιολόγησης των καταστάσεων ή εναλλακτικά ενός «ευρετικού μηχανισμού». Αποτέλεσμα αυτής της αξιολόγησης θα πρέπει να είναι η απόδοση μιας συγκεκριμένης τιμής στην κάθε μία από αυτές, ώστε, τελικά, να μπορούν να διαταχθούν.

Για το λόγο αυτό, στους αλγορίθμους BestFS και A* εισάγεται η έννοια της «ευριστικής συνάρτησης». Στο πλαίσιο της παρούσης εργασίας, ως ευρετικός μηχανισμός (ευριστική συνάρτηση) υιοθετήθηκε η *απόσταση Manhattan* των πλακιδίων από την τρέχουσα θέση τους στο πλαίσιο μέχρι τη σωστή τους θέση, με βάση την κατάσταση-στόχο. Η ευριστική συνάρτηση αυτή είναι κοινή και για τους δύο αλγορίθμους ευριστικής αναζήτησης.

Ωστόσο, οι δύο αυτοί αλγόριθμοι διαφέρουν ως προς τον τρόπο που διαχειρίζονται την απόσταση Manhattan ως μετρική που εφαρμόζεται επί μιας κατάστασης του προβλήματος. Πιο συγκεκριμένα:

- η τιμή της αξιολόγησης μιας κατάστασης προκύπτει αποκλειστικά από την ευριστική συνάρτηση, στην περίπτωση του BestFS
- η αντίστοιχη τιμή προκύπτει ως συνδυασμός (άθροισμα) της ευριστικής συνάρτησης και του βάθους της κατάστασης, στην περίπτωση του A*

Δηλαδή, ισχύει:

$$heuristicValue(aState) = manhattanDistance(aState), \text{ for BestFS}$$

ενώ,

$$heuristicValue(aState) = manhattanDistance(aState) + depth(aState), \text{ for A*}$$

3.4 Εφαρμογή των αλγορίθμων αναζήτησης σε C++

Μέχρι και αυτό το σημείο, έχουμε αναλύσει το αφαιρετικό πρίσμα υπό το οποίο μελετήθηκαν θεωρητικά οι τέσσερις αλγόριθμοι αναζήτησης. Στην ενότητα αυτή, θα προσπαθήσουμε να παρουσιάσουμε την αντικειμενοστραφή προοπτική, υπό την οποία οι αλγόριθμοι αυτοί υλοποιήθηκαν σε γλώσσα προγραμματισμού C++.

Βασικό στοιχείο αυτής της προοπτικής ήταν η αντιμετώπιση των αλγορίθμων αναζήτησης ως «εννοιών» του συστήματός μας, δηλαδή ως κλάσεων της C++. Λαμβάνοντας υπόψιν την θεωρητική ανάλυση των αλγορίθμων, κρίθηκε σκόπιμο να δημιουργηθεί μια γενική, αφηρημένη κλάση, η οποία αναπαριστά την έννοια του αλγορίθμου αναζήτησης.

Η κλάση αυτή ονομάστηκε `SearchAlgorithm` και αποτελεί μια (υψηλού επιπέδου) αναπαράσταση όλων των γενικών χαρακτηριστικών κορμού, που είναι κοινά μεταξύ των διαφορετικών αλγορίθμων. Πρακτικά, η διάρθρωσή της αφηρημένης αυτής κλάσης ακολουθεί πιστά τον γενικό αλγόριθμο αναζήτησης, όπως παρουσιάστηκε στις προηγούμενες ενότητες.

Κατόπιν, με κεντρικό πυλώνα την αφηρημένη αυτή κλάση, αναπτύχθηκαν τέσσερις επιπλέον επιμέρους κλάσεις, μια για κάθε έναν από τους αλγορίθμους αναζήτησης που μελετήθηκαν, με ενδεικτικά ονόματα. Κατά τη διαδικασία αυτή, αξιοποιήθηκε σε μεγάλο βαθμό το χαρακτηριστικό της κληρονομικότητας που η γλώσσα προσφέρει.

Πιο συγκεκριμένα, κάθε ένας από τους τέσσερις αλγορίθμους αναπτύχθηκε ως υπο-κλάση της αφηρημένης, κληρονομώντας από αυτή το σκαρίφημα και τις βασικές ιδιότητες της αναζήτησης. Με βάση αυτά, η κάθε κλάση-αλγόριθμος παρέχει με τη μορφή υπερκάλυψης συναρτήσεων τις εξειδικεύσεις επί του γενικού πλαισίου αναζήτησης, με πλήρη πιστότητα στα όσα αναλύθηκαν έως τώρα.

Εμβαθύνοντας λίγο περισσότερο στις σχέσεις κληρονομικότητας μεταξύ των αλγορίθμων, κρίνεται σκόπιμο να επισημανθεί πως η ανάπτυξη των κλάσεων έγινε σε δύο επίπεδα κληρονομικότητας. Το πρώτο επίπεδο αποτελείται από τις κλάσεις οι οποίες κληρονομούν απευθείας την αφηρημένη και περιλαμβάνει τις κλάσεις για τους αλγορίθμους DFS, BFS και BestFS. Το δεύτερο επίπεδο κληρονομεί εμμέσως (όχι απευθείας) την αφηρημένη και περιλαμβάνει την κλάση για τον A* αλγόριθμο.

Βασικός λόγος της επιλογής της πολυ-επίπεδης κληρονομικότητας είναι το γεγονός πως οι αλγόριθμοι BestFS και A* παρουσιάζουν πολλά κοινά στοιχεία και ελάχιστες διαφοροποιήσεις, και μάλιστα σε μεγαλύτερο βαθμό από ό,τι οποιοδήποτε άλλο ζεύγος αλγορίθμων. Για την ακρίβεια, η μόνη διαφοροποίηση μεταξύ των δύο αλγορίθμων είναι ο τρόπος προσδιορισμού της ευριστικής τιμής μιας κατάστασης, ενώ τα υπόλοιπα στοιχεία είναι κοινά.

Η σχέση κληρονομικότητας μεταξύ των κλάσεων αλγορίθμων αναζήτησης μπορεί να περιγραφεί από το ακόλουθο UML διάγραμμα:

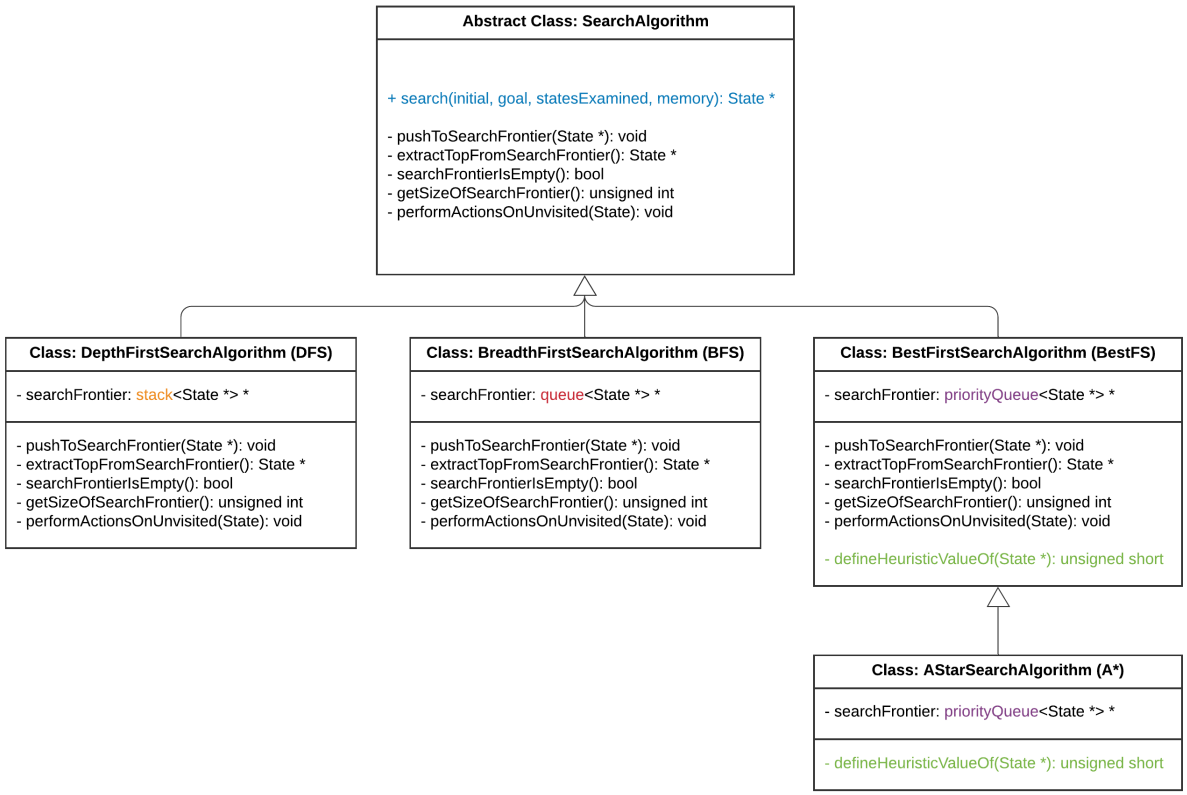


Figure 3: UML διάγραμμα για τις κλάσεις που αναπαριστούν τους αλγορίθμους αναζήτησης του συστήματος

4 Στατιστικά στοιχεία εκτέλεσης του συστήματος

Έχοντας αναλύσει σε θεωρητικό και πρακτικό επίπεδό τόσο τη μοντελοποίηση του προβλήματος όσο και τους αλγορίθμους αναζήτησης, το σύστημά μας είναι πλέον έτοιμο. Αυτός ήταν εξάλλου και ο απώτερος στόχος μας: η κατασκευή ενός συστήματος που θα μπορεί να επιλύει ένα τυχαίο δοθέν N-puzzle. Στο κεφάλαιο αυτό θα προσπαθήσουμε να συγκρίνουμε τους αλγορίθμους αναζήτησης μεταξύ τους ως προς διάφορα κριτήρια.

Πιο συγκεκριμένα, βασική ιδέα της προσέγγισής μας ήταν η διαμόρφωση ενός, όσο το δυνατόν, ενιαίου και αντιπροσωπευτικού πλαισίου εκτέλεσης του συστήματος. Με τον τρόπο αυτό, θα μπορούσαμε να μετρήσουμε επί πραγματικών δεδομένων την απόδοση των διαφορετικών αλγορίθμων αναζήτησης, εξάγοντας με πειραματικό τρόπο συμπεράσματα για αυτούς.

Για το σκοπό αυτό, το σύστημά μας εκτελέστηκε έναν συγκεκριμένο αριθμό φορών, καλούμενο, κάθε φορά, να επιλύσει ένα τυχαία διαμορφωμένο 8-puzzle. Συνολικά , εκτελέστηκαν 3.000 προσπάθειες επίλυσης τυχαίων puzzle με 8+1 πλακίδια. Εξ΄ αυτών, 1.500 αρχικές καταστάσεις (διαμορφωμένες τυχαία) ήταν επιλύσιμες, οι οποίες και καταγράφηκαν, ενώ οι υπόλοιπες ήταν μη επιλύσιμες. Για κάθε μία από τις επιλύσιμες καταστάσεις, εκτελέστηκαν και οι τέσσερις αλγόριθμοι αναζήτησης, ενώ καταγράφηκαν τα εξής στοιχεία:

- το πλήθος καταστάσεων που χρειάστηκε να ελεγχθούν, έως ότου βρεθεί λύση
- ο όγκος μνήμης που χρησιμοποιήθηκε για την εύρεση της λύσης
- το πλήθος ενεργειών από την αρχική κατάσταση μέχρι τη λύση που βρέθηκε, δηλαδή το βάθος της τελικής λύσης

Είναι σημαντικό να επισημανθεί πως τα στοιχεία αυτά καταγράφηκαν για κάθε μία από τις 1.500 επιλύσιμες τυχαίες αρχικές καταστάσεις και για κάθε έναν από τους τέσσερις διαφορετικούς αλγορίθμους που υποστηρίζονται από το σύστημά μας. Κατόπιν, τα στοιχεία αυτά υποβλήθηκαν σε στατιστική επεξεργασία.

Ειδικότερα, από τις 1.500 αρχικές καταστάσεις, λήφθηκαν υπόψιν μόνο οι μοναδικές μεταξύ τους, οι οποίες βρέθηκαν να είναι 1.140. Οι τελευταίες αποτέλεσαν και το τελικό δείγμα, από το οποίο προέκυψε η στατιστική ανάλυση που παρουσιάζεται στο παρόν κεφάλαιο.

4.1 Σύγκριση ως προς το πλήθος των ελεγχόμενων καταστάσεων

Το πρώτο κριτήριο σύγκρισης μεταξύ των αλγορίθμων αναζήτησης είναι ο αριθμός των καταστάσεων που χρειάστηκε να ελεγχθούν, έως ότου βρεθεί μία λύση του puzzle. Στο σημείο αυτό να διευκρινίσουμε ότι ως λύση θεωρούμε την ακολουθία κινήσεων που απαιτούνται προκειμένου να οδηγηθούμε από την δοθείσα αρχική κατάσταση στην κατάσταση-στόχο. Στον ακόλουθο πίνακα σημειώνεται ο ελάχιστος, μέσος και μέγιστος αριθμός καταστάσεων που χρειάστηκε να ελεγχθούν ανά εκτέλεση, για να βρεθεί μια λύση.

	DFS	**BFS**	**BestFS**	**A***
Ελάχιστο	198	211	12	16
Μέσο	64.860	92.019	294	1.716
Μέγιστο	150.672	181.403	1.103	18.036

Table 1: Ελάχιστο, μέσο και μέγιστο πλήθος ελεγχόμενων καταστάσεων ανά εκτέλεση του συστήματος

Ιδιαίτερο ενδιαφέρον παρουσιάζει και η κατανομή του πλήθους αυτού στα 1.140 puzzle που επιλύθηκαν από το σύστημα. Στα ιστογράμματα που ακολουθούν, το εύρος τιμών από την ελάχιστη έως και τη μέγιστη τιμή διαιρέθηκαν σε 100 ισομεγέθεις κλάσεις. Στα ιστογράμματα αυτά απεικονίζεται η συχνότητα του αριθμού των ελεγχόμενων καταστάσεων και για τους τέσσερις αλγορίθμους αναζήτησης: (από αριστερά προς τα δεξιά και από επάνω προς τα κάτω) DFS, BFS, BestFS και A*.

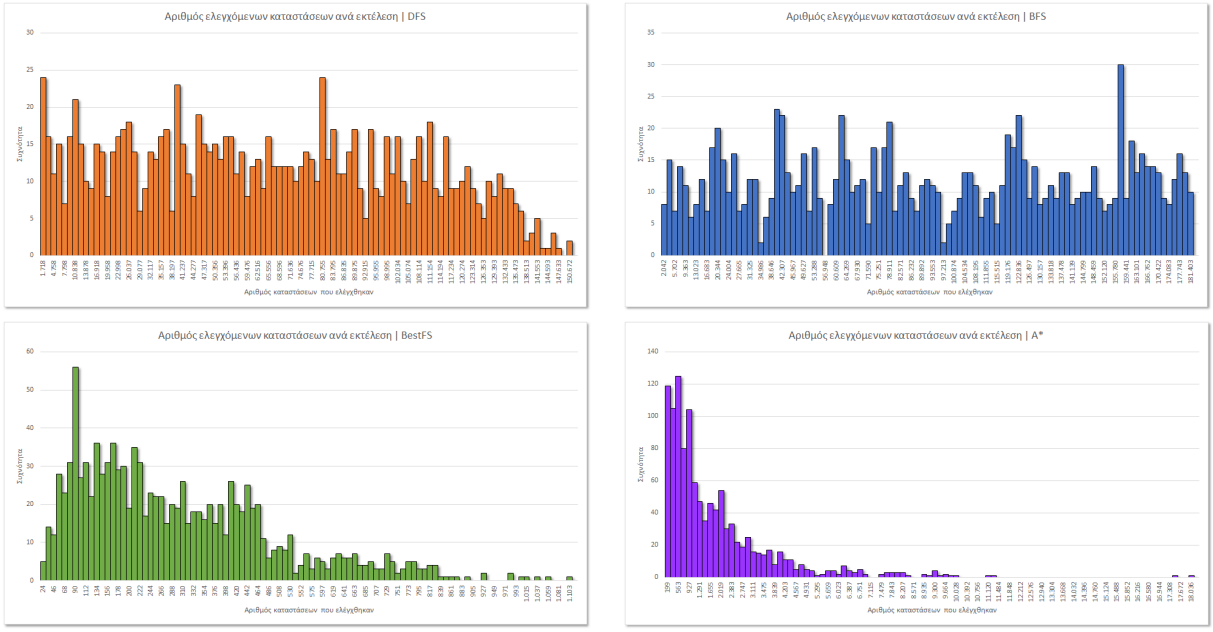


Table 2: Κατανομή πλήθους ελεγχόμενων καταστάσεων για τους τέσσερις αλγορίθμους αναζήτησης

Είναι εμφανές, τόσο από το εύρος τιμών όσο και από τα ιστογράμματα κατανομής πώς οι αλγόριθμοι ευρετικής αναζήτησης έχουν αισθητά καλύτερη απόδοση από τους τυφλούς αλγορίθμους.

Το συμπέρασμα αυτό είναι μάλλον αναμενόμενο, λαμβάνοντας υπόψιν τον τρόπο με τον οποίο γίνεται η αναζήτηση της λύσης στην κάθε μία κατηγορία. Η αξιοποίηση του ευρετικού μηχανισμού (απόσταση Manhattan των πλακιδίων) φαίνεται πως δίνει σημαντικό πλεονέκτημα στους ευρετικούς αλγορίθμους έναντι των τυφλών, οι οποίοι δεν διαθέτουν μηχανισμό αξιολόγησης καταστάσεων.

Σε επίπεδο κατηγοριών, τώρα, ο DFS φαίνεται να υπερτερεί έναντι του BFS, στην κατηγορία των τυφλών αλγορίθμων. Μια τέτοια διαπίστωση μπορεί να τεκμηριωθεί και από τα ιδιαίτερα χαρακτηριστικά των δύο αλγορίθμων. Αυτό σημαίνει πως η κατά βάθος διάτρεξη του γράφου καταστάσεων από τον DFS τον οδηγεί γρηγορότερα σε μία λύση, από την οπτική του αριθμού των ελεγχόμενων καταστάσεων πάντα. Αντίθετα, ο BFS επιλέγει να επεκτείνει πλήρως κάθε μία κατάσταση που ανακαλύπτει, γεγονός που στοιχίζει σε πλήθος ελέγχων.

Από την άλλη, στην κατηγορία των αλγορίθμων ευρετικής αξιολόγησης, η σύγκριση των ιστογραμμάτων οδηγεί σε ορισμένα ιδιαίτερα ενδιαφέροντα συμπεράσματα. Με βάση, λοιπόν, τα πειραματικά δεδομένα, ο BestFS φαίνεται να αποδίδει καλύτερα έναντι του A*. Θα μπορούσαμε, επομένως, να ισχυριστούμε πως η συμπερίληψη του βάθους μιας κατάστασης στο μηχανισμό αξιολόγησής δεν έχει τα προσδοκώμενα αποτελέσματα, αφού καθυστερεί (αποπροσανατολίζει) σημαντικά την εύρεση λύσης.

4.2 Σύγκριση ως προς τον όγκο της χρησιμοποιούμενης μνήμης

Ένα άλλο κριτήριο σύγκρισης μεταξύ των αλγορίθμων αναζήτησης είναι το πλήθος των μονάδων μνήμης που χρησιμοποιήθηκαν, έως ότου βρεθεί μια λύση του προβλήματος. Οι απαιτήσεις του κάθε αλγορίθμου σε μνήμη είναι μία παράμετρος με εξέχουσα σημασία στην αξιολόγηση αυτών. Μάλιστα, πρόκειται για μια μετρική που μπορεί ενδεικτικά να καθορίσει την εφαρμοσιμότητα ή μη ενός αλγορίθμου, όσο το πρόβλημα του N-puzzle κλιμακώνεται (πχ αύξηση του αριθμού N των πλακιδίων).

Ο ακόλουθος πίνακας περιέχει σημειωμένους τον ελάχιστο, μέσο και μέγιστο αριθμό μονάδων μνήμης που χρησιμοποιήθηκαν από τον κάθε αλγόριθμο ανά εκτέλεση, έως ότου βρεθεί μια λύση.

	DFS	BFS	BestFS	A*
Ελάχιστο	349	348	21	30
Μέσο	106.113	113.461	498	2.754
Μέγιστο	219.741	184.230	1.848	28.209

Table 3: Ελάχιστο, μέσο και μέγιστο πλήθος χρησιμοποιούμενων μονάδων μνήμης ανά εκτέλεση του συστήματος

Όπως και στην περίπτωση του πλήθους των ελεγχόμενων καταστάσεων, έτσι και στον όγκο χρησιμοποιούμενης μνήμης τα ιστογράμματα συχνοτήτων προσφέρουν μια εποπτική εικόνα για την κατανομή των δεδομένων. Και πάλι οι κλάσεις που χρησιμοποιήθηκαν για την αναπαράσταση ήταν 100, ενώ ίδια παραμένει και η σειρά παρουσίασης: (από αριστερά προς τα δεξιά και από επάνω προς τα κάτω) DFS, BFS, BestFS και A*.

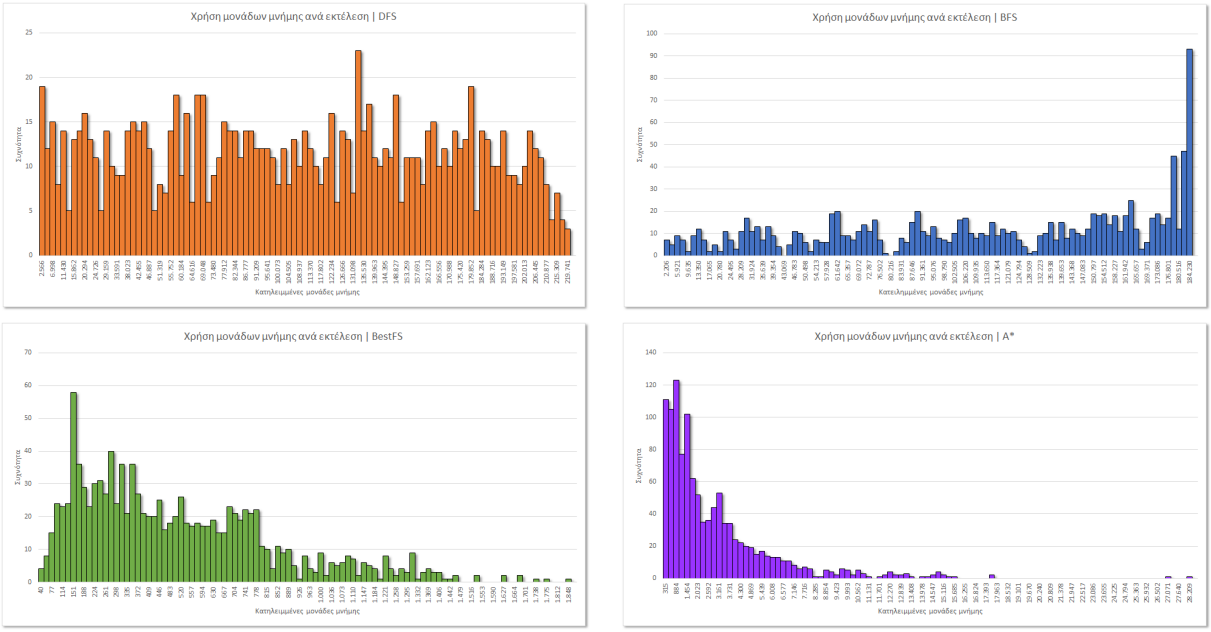


Table 4: Κατανομή πλήθους χρησιμοποιούμενων μονάδων μνήμης για τους τέσσερις αλγορίθμους αναζήτησης

Και σε αυτή την περίπτωση, οι αλγόριθμοι ευρετικής αναζήτησης παρουσιάζουν σημαντικά καλύτερη επίδοση από τους αντίστοιχους τυφλούς. Ο αλγόριθμος BestFS συγκεντρώνει εκ νέου τους καλύτερους δείκτες, αφήνοντας δεύτερο τον A*. Τέλος, οι αλγόριθμοι DFS και BFS έχουν υπερβολικά μεγάλες απαιτήσεις σε μνήμη, γεγονός που τους καθιστά πρακτικά ανεφάρμοστους για την επίλυση μεγαλύτερου μεγέθους puzzle. Ο πρώτος, βέβαια, παρουσιάζεται ελαφρώς καλύτερος, χωρίς αυτό να θεωρείται ικανοποιητικό.

4.3 Σύγκριση ως προς το βάθος της ευρισκόμενης λύσης

Το τρίτο και τελευταίο κριτήριο σύγκρισης των αλγορίθμων αναζήτησης είναι το βάθος της τελικής λύσης που ο κάθε ένας από αυτούς ανακαλύπτει. Πιο συγκεκριμένα, το βάθος της λύσης αντιστοιχεί στον αριθμό των κινήσεων που θα έπρεπε να κάνει ένας νοητός παίκτης, προκειμένου να μεταβεί από την αρχική δοθείσα κατάσταση στην κατάσταση-στόχο.

Σε ένα σύστημα Τεχνητής Νοημοσύνης, το βάθος της ευρισκόμενης λύσης μπορεί να διαδραματίζει καθοριστικό ρόλο στην απόδοση μιας εφαρμογής του πραγματικού κόσμου. Η «ευθύτητα» και «συντομία» της διαδρομής είναι τις περισσότερες φορές εξίσου σημαντική (αν όχι σημαντικότερη) από την εύρεση της ίδιας της κατάστασης-στόχου.

Η διαδικασία περιήγησης στα στατιστικά στοιχεία είναι πλέον γνωστή και εφαρμόζεται και στο κριτήριο του βάθους. Αρχικά, παρουσιάζονται ο ελάχιστος, μέσος και μέγιστος αριθμός κινήσεων της ευρισκόμενης λύσης

για τους τέσσερις αλγορίθμους:

	DFS	**BFS**	**BestFS**	**A***
Ελάχιστο	114	8	8	8
Μέσο	30.099	22	65	22
Μέγιστο	65.448	30	191	30

Table 5: Ελάχιστο, μέσο και μέγιστο βάθος ευρισκόμενης λύσης ανά εκτέλεση του συστήματος

Σε αυτή την περίπτωση συναντούμε ένα διαφορετικό μοτίβο, ως προς την αξιολόγηση των αλγορίθμων. Πιο συγκεκριμένα, δεν μπορεί να εξαχθεί γενικευμένο συμπέρασμα για την κατηγορία των τυφλών αλγορίθμων, μιας και οι DFS και BFS δεν παρουσιάζουν παρόμοια συμπεριφορά. Μπορεί ο πρώτος έχει άσχημους δείκτες, ωστόσο ο δεύτερος έχει εφάμιλλα καλούς με τους υπόλοιπους δύο, από την κατηγορία των ευριστικών.

Το σημείο αυτό είναι το πλέον κατάλληλο προκειμένου να εισάγουμε τα ιστογράμματα κατανομής, με στόχο αυτά να διαφωτίσουν περισσότερα στοιχεία για την επίδοση των αλγορίθμων. Ιδιαίτερη έμφαση, βέβαια, δίνεται στους αλγορίθμους BFS, BestFS και A*, οι οποίοι με τα μέχρι στιγμής δεδομένα εμφανίζονται να είναι αρκετά κοντά σε απόδοση, ως προς αυτή τη μετρική.

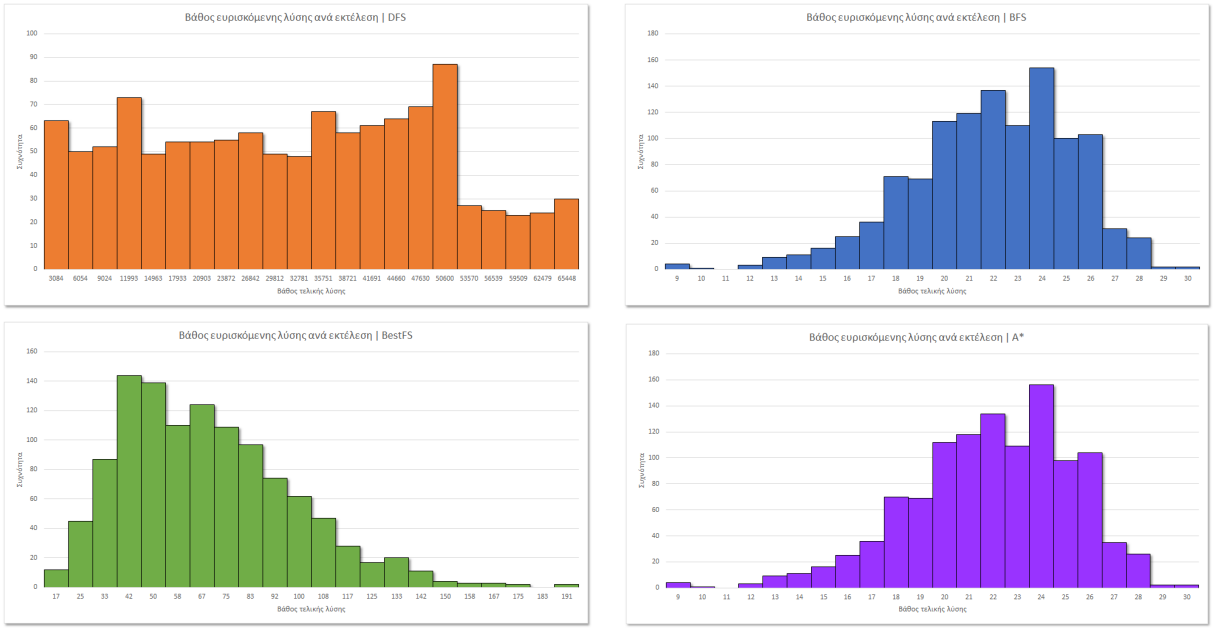


Table 6: Κατανομή βάθους ευρισκόμενης λύσης για τους τέσσερις αλγορίθμους αναζήτησης

Τα ιστογράμματα επιβεβαιώνουν τους θεωρητικούς μας ισχυρισμούς για τους αλγορίθμους. Έτσι, ο DFS είναι μακράν ο λιγότερο αξιόπιστος αλγόριθμος, αναφορικά με το βάθος της τελικής ευρισκόμενης λύσης. Η διαπίστωση αυτή είναι μάλλον αναμενόμενη, αν αναλογιστεί κανείς πως η επέκταση που πραγματοποιείται από τον αλγόριθμο είναι πρώτα κατά βάθος. Ως αποτέλεσμα, ο αλγόριθμος εξερευνά μια στενή και βαθιά περιοχή του γράφου καταστάσεων, βρίσκοντας μια λύση, η οποία, στην μεγάλη πλειονότητα των περιπτώσεων, απέχει αρκετά από τη βέλτιστη.

Επόμενος αλγόριθμος σε αυτή την αύξουσα κατάταξη επίδοσης (με μεγάλη διαφορά από τον DFS) είναι ο BestFS. Η αλήθεια είναι πως ο αλγόριθμος καταφέρνει να βρει τη βέλτιστη λύση ένα σημαντικά μεγάλο ποσοστό των εκτελέσεων, ενώ, τις υπόλοιπες, η λύση που βρίσκει δεν απέχει πολύ από τη βέλτιστη. Δεν θα πρέπει να ξεχνάμε πως πρόκειται για τον αλγόριθμο που αναδείχθηκε καλύτερος στις δύο προηγούμενες κατηγορίες σύγκρισης. Συνδυαστικά, επομένως, ο BestFS είναι μια ιδιαίτερα αξιόπιστη λύση, καθώς εξετάζει

λιγότερες καταστάσεις από τους υπολοίπους, έχει λιγότερες απαιτήσεις σε μνήμη και είναι αρκετά κοντά στη βέλτιστη λύση στη μεγάλη πλειονότητα των εκτελέσεων.

Σχεδόν αντίθετα είναι τα δεδομένα που ισχύουν για τον BFS. Ο αλγόριθμος βρίσκει πάντοτε τη βέλτιστη λύση, όπως φαίνεται και στα ιστογράμματα, ωστόσο έχει ιδιαίτερα μεγάλες απαιτήσεις σε μνήμη και επενεργεί μεγάλο αριθμό συγκρίσεων μέχρι την εύρεση της λύσης. Συνδυαστικά, επομένως, πρόκειται για έναν αλγόριθμο που θα μπορούσε να εφαρμοστεί σε συνθήκες όπου ο χρόνος που έχει στη διάθεσή του το σύστημα για να βρει τη λύση είναι μεγάλος, και το υπολογιστικό σύστημα στο οποίο εκτελείται έχει μεγάλη υπολογιστική ισχύ και μνήμη. Υπό αυτές τις συνθήκες, ο αλγόριθμος εγγυάται την εύρεση της βέλτιστης λύσης, ωστόσο η εξασφάλιση των συνθηκών αυτών είναι σχεδόν ουτοπική σε εφαρμογές πραγματικού χρόνου.

Τέλος, ο αλγόριθμος A* βρίσκει και αυτός πάντοτε τη βέλτιστη λύση, αναφορικά με το πλήθος των απαιτούμενων ενεργειών. Σε συνδυασμό με την καλή (αλλά όχι άριστη) εικόνα του στις δύο προηγούμενες μετρικές, ο αλγόριθμος φαίνεται να είναι η ενδεδειγμένη λύση σε περιπτώσεις όπου η βελτιστότητα (συντομία) της λύσης είναι ύψιστης σημασίας. Σε αυτή την περίπτωση, ωστόσο, θα πρέπει να είμαστε σε θέση να «πληρώσουμε» το τίμημα σε μνήμη και αριθμό ελεγχόμενων καταστάσεων. Τώρα, αν το τίμημα αυτό είναι αμελητέο ή απαγορευτικό εξαρτάται άμεσα από το πρόβλημα που καλούμαστε να λύσουμε, το διαθέσιμο χρόνο και τις δυνατότητες του υπολογιστικού μας συστήματος.