ΠΑΝΕΠΙΣΤΗΜΙΟ ΜΑΚΕΔΟΝΙΑΣ
ΣΧΟΛΗ ΕΠΙΣΤΗΜΩΝ ΠΛΗΡΟΦΟΡΙΑΣ
ΤΜΗΜΑ ΕΦΑΡΜΟΣΜΕΝΗΣ ΠΛΗΡΟΦΟΡΙΚΗΣ

Συγκριτική Μελέτη Domain-Dependent και Domain-Independent Planners για το πρόβλημα 'Rover' (IPC 2023)

Πτυχιακή Εργασία

του

Νικόλαου Τσαρίδη

Θεσσαλονίκη, Ιούλιος 2025

Συγκριτική Μελέτη Domain-Dependent και Domain-Independent Planners για το πρόβλημα 'Rover' (IPC 2023)

Νικόλαος Τσαρίδης

Προπτυχιακός Φοιτητής του Τμήματος Εφαρμοσμένης Πληροφορικής του Πανεπιστημίου Μακεδονίας

Πτυχιακή Εργασία

υποβάλλεται για τη μερική εκπλήρωση των απαιτήσεών του

Πτυχίου στην Εφαρμοσμένη Πληροφορική - Επιστήμης και Τεχνολογίας Υπολογιστών

Επιβλέπων καθηγητής
Ιωάννης Ρεφανίδης

Περίληψη

Ο τομέας του Αυτοματοποιημένου Σχεδιασμού χαρακτηρίζεται από έναν θεμελιώδη συμβιβασμό: αυτόν μεταξύ των ευέλικτων planners γενικής χρήσης (domain-independent), που προσαρμόζονται σε οποιοδήποτε πρόβλημα, και των εξειδικευμένων, υψηλής απόδοσης planners (domain-dependent), που είναι σχεδιασμένοι για ένα συγκεκριμένο πεδίο. Η παρούσα πτυχιακή εργασία στοχεύει στη διερεύνηση αυτού του χάσματος απόδοσης, εστιάζοντας στο απαιτητικό αριθμητικό πεδίο 'Rover' του Διεθνούς Διαγωνισμού Σχεδιασμού (IPC) του 2023.

Για τις ανάγκες της έρευνας, σχεδιάστηκε και υλοποιήθηκε σε γλώσσα C ένας πλήρως λειτουργικός, εξειδικευμένος planner, του οποίου η ισχύς πηγάζει από την ανάπτυξη μιας πρωτότυπης, εξελικτικής σειράς πέντε χειροποίητων ευρετικών συναρτήσεων. Οι ευρετικές αυτές, με διαφορετικές ιδιότητες παραδεκτότητας και πληροφοριακότητας, ενσωματώνουν προοδευτικά βαθύτερη γνώση της δομής και των προκλήσεων του προβλήματος 'Rover'.

Η πειραματική διαδικασία περιελάμβανε τόσο την εσωτερική αξιολόγηση των διαφορετικών διαμορφώσεων του planner, όσο και τη συγκριτική του ανάλυση έναντι των κορυφαίων συστημάτων που συμμετείχαν στον διαγωνισμό. Η μελέτη εξετάζει την απόδοση των διαφορετικών προσεγγίσεων υπό το πρίσμα των δύο βασικών κατηγοριών σχεδιασμού: της ταχείας εύρεσης μιας ικανοποιητικής λύσης (satisficing) και της εγγυημένης εύρεσης της βέλτιστης (optimal), αναδεικνύοντας τη σημασία της εξειδικευμένης γνώσης στον τομέα.

Λέξεις-κλειδιά: Αυτόματος Σχεδιασμός, Ευρετική Αναζήτηση, Ευρετική Συνάρτηση, Βέλτιστη Σχεδίαση, Αριθμητικός Σχεδιασμός, Domain-Dependent Planners, Διεθνής Διαγωνισμός Σχεδιασμού (IPC).

Abstract

The field of Automated Planning is characterized by a fundamental trade-off: that between flexible, general-purpose (domain-independent) planners, which can be adapted to any problem, and specialized, high-performance (domain-dependent) planners, which are designed for a specific field. This thesis aims to investigate this performance gap, focusing on the demanding numerical domain 'Rover' of the 2023 International Planning Competition (IPC).

For the purposes of the research, a fully functional, specialized planner was designed and implemented in C language, whose power stems from the development of an original, evolutionary series of five handmade heuristic functions. These heuristics, with different acceptability and informativeness properties, progressively incorporate deeper knowledge of the structure and challenges of the Rover problem.

The experimental process included both an internal evaluation of the different configurations of the planner and a comparative analysis with the leading systems participating in the competition. The study examines the performance of different approaches in light of two basic design categories: rapidly finding a satisfactory solution (satisficing) and guaranteed finding the optimal solution, highlighting the importance of specialized knowledge in the field.

Keywords: Automated Planning, Heuristic Search, Heuristic Function, Optimal Planning, Numerical Planning, Domain-Dependent Planners, International Planning Competition (IPC).

Ευχαριστίες

Με την ολοκλήρωση της παρούσας πτυχιακής εργασίας, θα ήθελα να εκφράσω τις θερμές μου ευχαριστίες σε όσους συνέβαλαν με τον τρόπο τους στην επιτυχή της έκβαση.

Πρωτίστως, αισθάνομαι την ανάγκη να ευχαριστήσω θερμά τον επιβλέποντα καθηγητή μου, κύριο Ιωάννη Ρεφανίδη, για την εμπιστοσύνη που μου έδειξε αναθέτοντάς μου το συγκεκριμένο θέμα. Η πολύτιμη καθοδήγηση, η αμέριστη επιστημονική του στήριξη και οι εύστοχες παρατηρήσεις του σε όλα τα στάδια της εκπόνησης υπήρξαν καθοριστικής σημασίας. Τον ευχαριστώ ιδιαιτέρως για την υπομονή και την προθυμία με την οποία απαντούσε σε κάθε μου ερώτημα, παρέχοντάς μου τα απαραίτητα εφόδια για να ανταπεξέλθω στις προκλήσεις της έρευνας.

Ιδιαίτερες ευχαριστίες οφείλω στην οικογένειά μου, για τη διαρκή, ανιδιοτελή και πολύπλευρη υποστήριξή της, τόσο ηθική όσο και ψυχολογική, καθ' όλη τη διάρκεια των σπουδών μου. Η κατανόηση που επέδειξαν, ειδικά κατά τις περιόδους εντατικής μελέτης, αποτέλεσε θεμέλιο λίθο για την προσπάθειά μου.

Τέλος, θα ήθελα να ευχαριστήσω τους φίλους μου, για την ψυχολογική τους στήριξη και την ενθάρρυνση. Με τις συζητήσεις και τις στιγμές χαλάρωσης που μοιραστήκαμε, έκαναν αυτό το ακαδημαϊκό ταξίδι μια σαφώς πιο ευχάριστη και δημιουργική εμπειρία.

Περιεχόμενα

Κατάλογος σχημάτων

Κατάλογος πινάκων

Κατάλογος Αλγορίθμων

Κεφάλαιο 1

Εισαγωγή

1.1 Automated Planning

Η Τεχνητή Νοημοσύνη (ΤΝ) επιδιώκει, στον πυρήνα της, τη δημιουργία ευφυών οντοτήτων, συστημάτων που είναι ικανά να αντιλαμβάνονται το περιβάλλον τους και να αναλαμβάνουν δράσεις για την επίτευξη στόχων. Μια θεμελιώδης προϋπόθεση για την επίτευξη αυτής της αυτονομίας είναι η ικανότητα λήψης αποφάσεων. Ο Αυτοματοποιημένος Σχεδιασμός (Automated Planning) αποτελεί ακριβώς τον κλάδο της ΤΝ που μελετά και υλοποιεί αυτή την ικανότητα, εστιάζοντας στην αυτόματη παραγωγή μιας ακολουθίας δράσεων –ενός πλάνου– που μετασχηματίζει τον κόσμο από μια αρχική σε μια επιθυμητή τελική κατάσταση [1, 2].

Οι εφαρμογές του Automated Planning διατρέχουν οριζόντια ένα ευρύ φάσμα τομέων. Στη ρομποτική, καθοδηγεί τη λήψη αποφάσεων για την πλοήγηση και τον χειρισμό αντικειμένων, ενώ στον τομέα των logistics βελτιστοποιεί πολύπλοκα προβλήματα δρομολόγησης και διαχείρισης πόρων[1]. Ένα εξέχον παράδειγμα της σημασίας του σε απαιτητικά περιβάλλοντα είναι οι αποστολές εξερεύνησης του διαστήματος, όπου αυτόνομα οχήματα, όπως τα rovers, βασίζονται σε αλγορίθμους σχεδιασμού για να εκτελέσουν τις αποστολές τους με ασφάλεια και αποδοτικότητα, διαχειριζόμενα κρίσιμους πόρους όπως η ενέργεια [3].

1.2 Numeric Planning και το Πρόβλημα 'Rover'

Ενώ η κλασική σχεδίαση έθεσε τις βάσεις, πολλά ρεαλιστικά προβλήματα εισάγουν μια επιπλέον διάσταση πολυπλοκότητας: τις αριθμητικές μεταβλητές. Αυτή η διαπίστωση οδήγησε στη διαμόρφωση του πεδίου του **Αριθμητικού Σχεδιασμού (Numeric Planning)**, όπου οι αποφάσεις και οι συνθήκες δεν είναι απλώς λογικές, αλλά εξαρτώνται από συνεχείς ή διακριτές τιμές, όπως η ενέργεια, ο χρόνος ή η χωρητικότητα. Η ανάγκη για την τυποποιημένη περιγραφή

τέτοιων προβλημάτων καλύφθηκε από τη γλώσσα PDDL2.1, η οποία εισήγαγε για πρώτη φορά την έννοια των αριθμητικών χαρακτηριστικών (numeric fluents) [4].

Το πεδίο "Rover" αποτελεί ένα εμβληματικό πρόβλημα-αναφοράς (benchmark) για τον αριθμητικό σχεδιασμό. Μοντελοποιώντας τις προκλήσεις που αντιμετωπίζει ένα πλανητικό όχημα εξερεύνησης, συνδυάζει λογικούς περιορισμούς (π.χ., η λήψη δείγματος απαιτεί την παρουσία του οχήματος στη σωστή τοποθεσία) με αριθμητικούς (π.χ., η μετακίνηση καταναλώνει ενέργεια, η οποία είναι πεπερασμένη). Η αξιολόγηση της απόδοσης των planners σε τέτοια απαιτητικά προβλήματα πραγματοποιείται στο πλαίσιο του **Διεθνούς Διαγωνισμού Σχεδιασμού (International Planning Competition - IPC)**. Ο διαγωνισμός αυτός αποτελεί την κορυφαία ακαδημαϊκή αρένα για τη σύγκριση των state-of-the-art συστημάτων [5]. Η διαχρονική παρουσία του "Rover" σε πολλαπλούς διαγωνισμούς υπογραμμίζει την αξία του ως ένα σταθερά δύσκολο και επίκαιρο πρόβλημα για την έρευνα στον σχεδιασμό.

1.3 Κίνητρο και Ερευνητική Υπόθεση

Η προσέγγιση για την επίλυση προβλημάτων σχεδιασμού ιστορικά ακολούθησε δύο διακριτές, σχεδόν αντίθετες, φιλοσοφίες. Από τη μία πλευρά, βρίσκεται η ανάπτυξη των **planners ανεξάρτητων από το πεδίο (domain-independent planners)**. Αυτά τα συστήματα είναι σχεδιασμένα για να είναι καθολικής εφαρμογής. Στόχος τους είναι να επιλύουν οποιοδήποτε πρόβλημα μπορεί να εκφραστεί σε μια τυποποιημένη γλώσσα, όπως η PDDL, χωρίς να απαιτούν καμία προηγούμενη γνώση για τις ιδιαιτερότητες του εκάστοτε πεδίου [6]. Η ευελιξία αυτή, ωστόσο, συχνά συνοδεύεται από ένα σημαντικό κόστος στην απόδοση, καθώς οι γενικοί αλγόριθμοι ενδέχεται να μην είναι σε θέση να εκμεταλλευτούν τις κρυφές δομικές ιδιότητες ή τις συντομεύσεις που υπάρχουν σε ένα συγκεκριμένο πρόβλημα.

Στον αντίποδα, βρίσκονται οι **planners εξαρτώμενοι από το πεδίο (domain-dependent planners)**. Αυτοί οι planners είναι «χτισμένοι στο χέρι» (hand-crafted) για ένα και μόνο συγκεκριμένο πρόβλημα. Ενσωματώνουν εξειδικευμένη γνώση για τη δυναμική και τους περιορισμούς του πεδίου, επιτρέποντάς τους να επιτυγχάνουν εξαιρετικά υψηλή απόδοση, τόσο σε ταχύτητα επίλυσης όσο και σε ποιότητα του παραγόμενου πλάνου. Το προφανές μειονέκτημά τους είναι η έλλειψη γενικότητας: απαιτούν σημαντική προσπάθεια και εξειδίκευση για την ανάπτυξή τους και είναι εντελώς άχρηστοι για οποιοδήποτε άλλο πρόβλημα πέραν αυτού για το οποίο σχεδιάστηκαν [7].

Αυτή η θεμελιώδης αντιπαράθεση μεταξύ γενικότητας και εξειδίκευσης θέτει το κεντρικό ερευνητικό ερώτημα που διερευνά η παρούσα πτυχιακή εργασία. Πόσο μεγάλο είναι τελικά το κενό στην απόδοση μεταξύ ενός state-of-the-art domain-independent planner και ενός προσεκτικά σχεδιασμένου, εξειδικευμένου

planner για ένα απαιτητικό αριθμητικό πρόβλημα όπως το "Rover"; Η ποσοτική απάντηση σε αυτό το ερώτημα δεν έχει μόνο θεωρητικό ενδιαφέρον, αλλά παρέχει και πρακτική γνώση σχετικά με τον συμβιβασμό (trade-off) που καλείται να κάνει ένας μηχανικός ή ερευνητής όταν επιλέγει την κατάλληλη προσέγγιση για την επίλυση ενός νέου προβλήματος σχεδιασμού.

1.4 Σκοπός και Συνεισφορά της Εργασίας

Με βάση το ερευνητικό ερώτημα που τέθηκε στην προηγούμενη ενότητα, ο κύριος σκοπός της παρούσας πτυχιακής εργασίας είναι διττός. Πρώτον, η σχεδίαση και υλοποίηση ενός πλήρως λειτουργικού, εξειδικευμένου (domain-dependent) planner για το πεδίο προβλημάτων "Rover", ο οποίος αξιοποιεί τους αλγορίθμους ευρετικής αναζήτησης πρώτα-στο-καλύτερο και A*. Δεύτερον, η συστηματική και ποσοτική αξιολόγηση της απόδοσης αυτού του planner σε σύγκριση με σύγχρονους, state-of-the-art planners γενικής χρήσης (domain-independent) που συμμετείχαν στον Διεθνή Διαγωνισμό Σχεδιασμού του 2023.

Για την επίτευξη του παραπάνω σκοπού, η έρευνα επικεντρώνεται στην απάντηση των ακόλουθων ερευνητικών ερωτημάτων:

1. Πόσο σημαντική είναι η διαφορά στην **ταχύτητα επίλυσης** μεταξύ του εξειδικευμένου planner και των συστημάτων γενικής χρήσης;
2. Είναι εφικτός ο σχεδιασμός νέων, **ισχυρών ευρετικών συναρτήσεων** που εκμεταλλεύονται τη δομή του πεδίου "Rover", και τι ιδιότητες (π.χ. παραδεκτότητα, συνέπεια) μπορούν να επιδείξουν;
3. Πώς συγκρίνεται **η ποιότητα των παραγόμενων πλάνων**, μετρούμενη συνήθως ως το συνολικό τους κόστος;

Η συνεισφορά της παρούσας εργασίας είναι τριπλή και εντοπίζεται στους εξής άξονες:

- **Υλοποίηση**: Η κατασκευή ενός αυτόνομου domain-dependent planner, ικανού να επιλύει προβλήματα του πεδίου "Rover".
- **Σχεδιασμός**: Ο σχεδιασμός και η παρουσίαση νέων ευρετικών συναρτήσεων προσαρμοσμένων στις ιδιαιτερότητες του προβλήματος, με ανάλυση των θεωρητικών τους ιδιοτήτων.
- **Ανάλυση**: Η εμπειρική ανάλυση και ποσοτικοποίηση του συμβιβασμού (trade-off) μεταξύ της ευελιξίας των γενικών συστημάτων και της αποδοτικότητας των εξειδικευμένων λύσεων, παρέχοντας απτά δεδομένα σε ένα σύγχρονο και απαιτητικό πρόβλημα.

1.5 Διάρθρωση της Εργασίας

Η παρούσα εργασία διαρθρώνεται σε πέντε κεφάλαια. Στο **Κεφάλαιο 2**, παρουσιάζεται το απαραίτητο θεωρητικό υπόβαθρο. Αναλύονται οι θεμελιώδεις αρχές των αλγορίθμων ευρετικής αναζήτησης, με έμφαση στον αλγόριθμο A*, καθώς και οι βασικές έννοιες του Αυτοματοποιημένου Σχεδιασμού. Το **Κεφάλαιο 3** εστιάζει στην αναλυτική περιγραφή του προβλήματος. Παρουσιάζεται το πλαίσιο του Διεθνούς Διαγωνισμού Σχεδιασμού (IPC) και εξετάζεται λεπτομερώς η δομή και οι ιδιαιτερότητες του πεδίου "Rover". Στο **Κεφάλαιο 4**, το οποίο αποτελεί τον πυρήνα της εργασίας, περιγράφεται η μεθοδολογία που ακολουθήθηκε. Παρουσιάζεται η υλοποίηση του εξειδικευμένου planner, αναλύονται οι ευρετικές συναρτήσεις που σχεδιάστηκαν και παρατίθενται τα πειραματικά αποτελέσματα και η συγκριτική τους ανάλυση. Τέλος, το **Κεφάλαιο 5** ολοκληρώνει την εργασία, συνοψίζοντας τα βασικά συμπεράσματα της έρευνας, και προτείνοντας κατευθύνσεις για μελλοντικές βελτιώσεις και περαιτέρω έρευνα.

Κεφάλαιο 2

Θεωρητικό Υπόβαθρο

2.1 Αλγόριθμοι Αναζήτησης σε Χώρους Καταστάσεων

Η διαδικασία επίλυσης ενός προβλήματος στην Τεχνητή Νοημοσύνη μπορεί συχνά να μοντελοποιηθεί ως η διαδικασία εύρεσης μιας λύσης μέσα σε έναν, συνήθως τεράστιο, χώρο πιθανών επιλογών. Το υπολογιστικό πλαίσιο που χρησιμοποιείται για την τυποποίηση και την αντιμετώπιση τέτοιων προβλημάτων είναι η **αναζήτηση σε χώρους καταστάσεων (state-space search)**. Η παρούσα ενότητα θέτει τις βάσεις για την κατανόηση των αλγορίθμων που αποτελούν τον πυρήνα κάθε σύγχρονου συστήματος σχεδιασμού. Ξεκινώντας από τον τυπικό ορισμό ενός προβλήματος αναζήτησης, η ανάλυση προχωρά στις αποδοτικές μεθόδους της ενημερωμένης (ή ευρετικής) αναζήτησης, και κορυφώνεται με την παρουσίαση του αλγορίθμου A*, ο οποίος αποτελεί τον ακρογωνιαίο λίθο για την υλοποίηση που περιγράφεται σε επόμενα κεφάλαια[1, 2].

2.1.1 Ορισμός του Προβλήματος Αναζήτησης

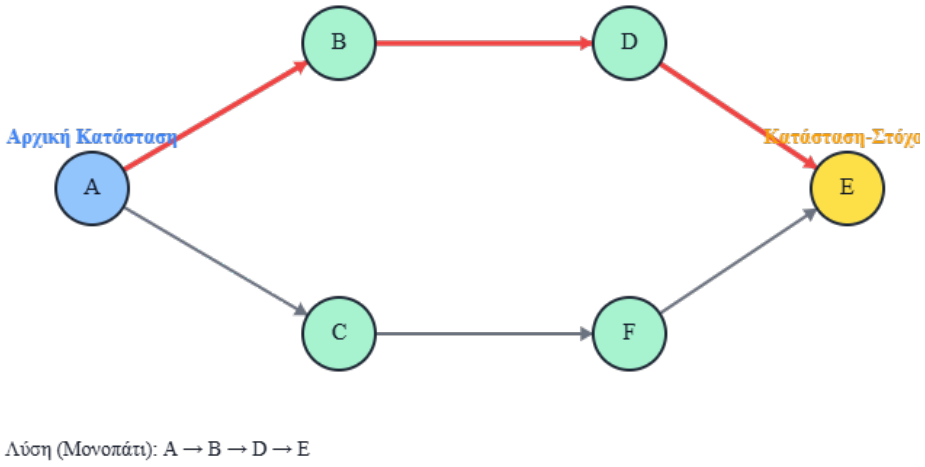
Για να επιλυθεί ένα πρόβλημα μέσω αλγορίθμων, το πρώτο και κρισιμότερο βήμα είναι ο ακριβής και τυπικός του ορισμός. Στο πλαίσιο της Τεχνητής Νοημοσύνης, αυτό επιτυγχάνεται με την αναγωγή του προβλήματος σε ένα πρόβλημα αναζήτησης, το οποίο ορίζεται τυπικά από πέντε στοιχεία:

1. Ο **χώρος καταστάσεων (state space)**, που περιλαμβάνει το σύνολο όλων των πιθανών καταστάσεων στις οποίες μπορεί να βρεθεί ο κόσμος.
2. Η **αρχική κατάσταση (initial state)**, από την οποία ξεκινά η διαδικασία επίλυσης.
3. Ένα σύνολο από **δράσεις (actions)** ή **τελεστές (operators)**, που είναι διαθέσιμες σε κάθε κατάσταση. Κάθε δράση, όταν εφαρμόζεται, οδηγεί σε μια νέα κατάσταση **μέσω ενός μοντέλου μετάβασης (transition model)**.

4. Μια **συνθήκη-στόχος (goal test)**, η οποία ελέγχει αν μια δεδομένη κατάσταση είναι η επιθυμητή τελική κατάσταση.

5. Μια **συνάρτηση κόστους μονοπατιού (path cost function)**, η οποία αποδίδει ένα αριθμητικό κόστος σε κάθε μονοπάτι. Στόχος της αναζήτησης είναι συνήθως η εύρεση του μονοπατιού με το ελάχιστο δυνατό κόστος, γνωστό και ως βέλτιστη λύση.

Στο πλαίσιο της παρούσας εργασίας, μια κατάσταση θα μπορούσε να είναι η ακριβής θέση ενός οχήματος Rover, το επίπεδο της ενέργειάς του και η κατάσταση των δειγμάτων που έχει συλλέξει. Μια δράση θα ήταν η εντολή *navigate* προς μια γειτονική τοποθεσία, και ο στόχος θα ήταν η ολοκλήρωση όλων των επιστημονικών αποστολών. Η δομή αυτή μπορεί να αναπαρασταθεί οπτικά ως ένας κατευθυνόμενος γράφος, όπου οι κόμβοι είναι οι καταστάσεις και οι ακμές οι δράσεις που τις συνδέουν (Σχήμα 2.1).



Σχήμα 2.1: Γραφική αναπαράσταση προβλήματος αναζήτησης.

2.1.2 Μη-Ενημερωμένοι και Ενημερωμένοι Αλγόριθμοι Αναζήτησης

Οι στρατηγικές αναζήτησης διακρίνονται σε δύο κύριες κατηγορίες, ανάλογα με το αν αξιοποιούν πληροφορία ειδική για το πρόβλημα. Οι **μη-ενημερωμένοι (uninformed)** ή "τυφλοί" (blind) αλγόριθμοι αναζήτησης δεν διαθέτουν καμία γνώση για την απόσταση ή την κατεύθυνση προς τον στόχο. Εξερευνούν τον

χώρο καταστάσεων συστηματικά, όπως για παράδειγμα η **Αναζήτηση Πρώτα-σε-Πλάτος (Breadth-First Search - BFS)**, η οποία επεκτείνει όλους τους κόμβους σε ένα δεδομένο βάθος πριν προχωρήσει στο επόμενο επίπεδο, ή η **Αναζήτηση Πρώτα-σε-Βάθος (Depth-First Search - DFS)**, η οποία ακολουθεί ένα μονοπάτι μέχρι το τέλος του πριν οπισθοδρομήσει. Παρότι είναι πλήρεις (complete) και, στην περίπτωση του BFS, βέλτιστοι για μοναδιαίο κόστος, η απόδοσή τους φθίνει δραματικά σε μεγάλους χώρους καταστάσεων λόγω της έλλειψης καθοδήγησης.

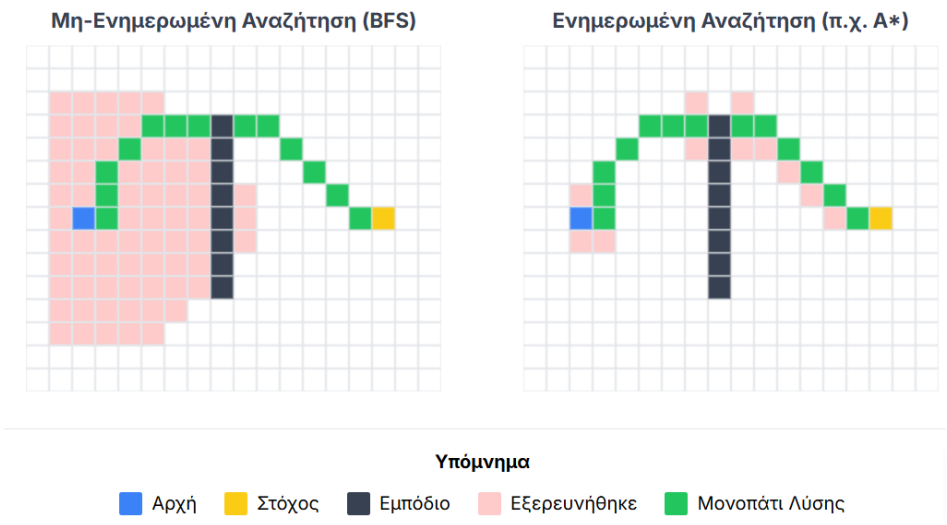
Στον αντίποδα των τυφλών μεθόδων, **οι ενημερωμένοι (informed)** ή **ευρετικοί (heuristic)** αλγόριθμοι αναζήτησης εκμεταλλεύονται πληροφορία ειδική για το πρόβλημα προκειμένου να καθοδηγήσουν την αναζήτηση προς τις πιο υποσχόμενες περιοχές του χώρου καταστάσεων. Αυτή η προσέγγιση, η οποία πλέον αποτελεί την κυρίαρχη μεθοδολογία στον αυτοματοποιημένο σχεδιασμό, ενσωματώνει τη γνώση μέσω μιας **ευρετικής συνάρτησης (heuristic function)**, που συμβολίζεται ως $h(n)$. Η συνάρτηση $h(n)$ δέχεται ως είσοδο έναν κόμβο n και επιστρέφει μια εκτίμηση του κόστους του φθηνότερου μονοπατιού από τον κόμβο n προς μια κατάσταση-στόχο [8]. Μια καλή ευρετική συνάρτηση μπορεί να μειώσει δραματικά την πολυπλοκότητα της αναζήτησης (Σχήμα 2.2).

Η γενική κατηγορία αλγορίθμων που αξιοποιεί τέτοιες συναρτήσεις ονομάζεται **Αναζήτηση Πρώτα-στο-Καλύτερο (Best-First Search)**. Οι αλγόριθμοι αυτοί διατηρούν μια λίστα κόμβων προς επέκταση (τη λίστα OPEN) και σε κάθε βήμα επιλέγουν να επεκτείνουν τον κόμβο που φαίνεται να είναι ο "καλύτερος" σύμφωνα με μια **συνάρτηση αξιολόγησης (evaluation function)**, $f(n)$. Ένα απλό παράδειγμα είναι ο άπληστος αλγόριθμος πρώτα-στο-καλύτερο (Greedy Best-First Search), ο οποίος χρησιμοποιεί $f(n) = h(n)$, προσπαθώντας να ελαχιστοποιήσει την εκτιμώμενη απόσταση προς τον στόχο, αγνοώντας το κόστος του μονοπατιού που έχει ήδη διανυθεί.

2.1.3 Ο Αλγόριθμος A*

Ο αλγόριθμος A (A-star)* είναι ο πιο διαδεδομένος και ευρέως χρησιμοποιούμενος αλγόριθμος ενημερωμένης αναζήτησης. Η δημοφιλία και η αποτελεσματικότητά του πηγάζουν από τον τρόπο που συνδυάζει τα πλεονεκτήματα δύο άλλων στρατηγικών: την προσοχή στο ήδη διανυθέν κόστος, όπως ο αλγόριθμος του Dijkstra, και την εστίαση στον στόχο, όπως ο άπληστος αλγόριθμος πρώτα-στο-καλύτερο. Ο A* παρουσιάστηκε για πρώτη φορά στο **θεμελιώδες άρθρο των Hart, Nilsson και Raphael το 1968** και παραμένει μέχρι σήμερα ο de facto αλγόριθμος για την εύρεση βέλτιστων μονοπατιών σε προβλήματα αναζήτησης [9].

Η ισχύς του A* βασίζεται στη συνάρτηση αξιολόγησης που χρησιμοποιεί για να επιλέξει τον επόμενο κόμβο προς επέκταση από τη λίστα των υποψήφιων κόμβων (OPEN list):



Σχήμα 2.2: Σύγκριση Μη-Ενημερωμένης και Ενημερωμένης Αναζήτησης.

$$f(n) = g(n) + h(n)$$

όπου:

- Το $g(n)$ είναι το **πραγματικό κόστος** του μονοπατιού από την αρχική κατάσταση μέχρι τον τρέχοντα κόμβο n.
- Το $h(n)$ είναι η **ευρετική εκτίμηση** του κόστους από τον κόμβο n μέχρι την κατάσταση-στόχο.

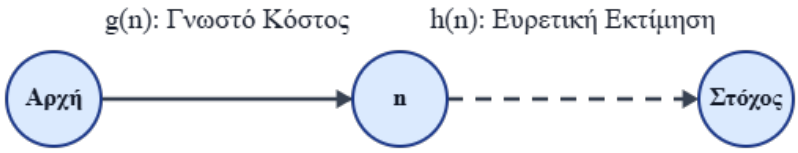
Επομένως, η f(n) αντιπροσωπεύει μια εκτίμηση του συνολικού κόστους του φθηνότερου μονοπατιού λύσης που διέρχεται από τον κόμβο η [1, 3]. Ο αλγόριθμος επιλέγει πάντα να επεκτείνει τον κόμβο με τη μικρότερη τιμή f, εξισορροπώντας έτσι το ήδη διανυθέν κόστος με το εκτιμώμενο μελλοντικό κόστος. Η επιτυχία του A* εξαρτάται σχεδόν αποκλειστικά από την ποιότητα της ευρετικής συνάρτησης $h(n)$. Όπως θα αναλυθεί στο Κεφάλαιο 4, ο σχεδιασμός μιας ισχυρής και ταυτόχρονα υπολογιστικά αποδοτικής ευρετικής για το πεδίο 'Rover' αποτελεί την καρδιά της υλοποίησης της παρούσας εργασίας.

Algorithm 1 Αλγόριθμος A*

```
1: function A_STAR(start_node, goal_node)
2:     OPEN_set ← {start_node}                ▷ Το σύνολο των κόμβων που έχουν ανακαλυφθεί αλλά δεν έχουν εξερευνηθεί.
3:     CLOSED_set ← {}  ▷ Το σύνολο των κόμβων που έχουν ήδη εξερευνηθεί.
4:     came_from ← an empty map      ▷ Λεξικό για την αποθήκευση του γονέα κάθε κόμβου.
5:     g_score ← map with default value of ∞
6:     g_score[start_node] ← 0 ▷ Κόστος από την αρχή μέχρι τον κόμβο (g_score).
7:     f_score ← map with default value of ∞    ▷ Εκτίμηση συνολικού κόστους από την αρχή στον στόχο (f_score).
8:     f_score[start_node] ← h(start_node)        ▷ h() είναι η ευρετική συνάρτηση.
9:     while OPEN_set is not empty do
10:        current_node ← a node in OPEN_set with the lowest f_score[]
11:        if current_node == goal_node then
12:            return reconstruct_path(came_from, current_node)
13:        end if
14:        OPEN_set.remove(current_node)
15:        CLOSED_set.add(current_node)
16:        for each neighbor of current_node do
17:            if neighbor in CLOSED_set then
18:                continue     ▷ Αγνοούμε τον γείτονα που έχει ήδη εξερευνηθεί.
19:            end if
20:            tentative_g_score ← g_score[current_node] + cost(current_node, neighbor)
21:            if neighbor not in OPEN_set then
22:                OPEN_set.add(neighbor)
23:            else if tentative_g_score ≥ g_score[neighbor] then
24:                continue                ▷ Έχει ήδη βρεθεί καλύτερο μονοπάτι.
25:            end if
26:            came_from[neighbor] ← current_node  ▷ Αυτό το μονοπάτι είναι το καλύτερο. Το καταγράφουμε.
27:            g_score[neighbor] ← tentative_g_score
28:            f_score[neighbor] ← g_score[neighbor] + h(neighbor)
29:        end for
30:    end while
31:    return failure                                ▷ Δεν βρέθηκε λύση.
32: end function
```

Algorithm 2 Ανακατασκευή Μονοπατιού

```
function RECONSTRUCT_PATH(came_from, current_node)
    total_path ← {current_node}
    while current_node in came_from do
        current_node ← came_from[current_node]
        total_path.prepend(current_node)
    end while
    return total_path
end function
```



Σχήμα 2.3: Οπτικοποίηση της συνάρτησης αξιολόγησης f(n) του A.*

2.1.4 Παράδειγμα Εκτέλεσης του A*

Για την καλύτερη κατανόηση του αλγορίθμου, θα εξετάσουμε την εκτέλεσή του βήμα-προς-βήμα σε ένα απλό πρόβλημα εύρεσης μονοπατιού, όπως αυτό που απεικονίζεται στο Σχήμα 2.4. Στόχος είναι η εύρεση του βέλτιστου μονοπατιού από τον κόμβο **S (Start)** στον κόμβο **G (Goal)**.

Ο αλγόριθμος χρησιμοποιεί δύο λίστες: την **OPEN**, μια ουρά προτεραιότητας με τους κόμβους προς εξερεύνηση, ταξινομημένη με βάση τη μικρότερη τιμή $f(n)$, και την **CLOSED**, ένα σύνολο με τους κόμβους που έχουν ήδη εξερευνηθεί.

1. **Αρχικοποίηση**: Ο αλγόριθμος ξεκινά με τον κόμβο S. Υπολογίζουμε το $f(S) = g(S) + h(S) = 0 + 5 = 5$.
   - **OPEN**: { `S(f=5)` }
   - **CLOSED**: { }
2. **Επέκταση του S**: Ο κόμβος με το μικρότερο f στην OPEN είναι ο S. Τον εξάγουμε και τον προσθέτουμε στην CLOSED. Εξετάζουμε τους γείτονές του: A και B.

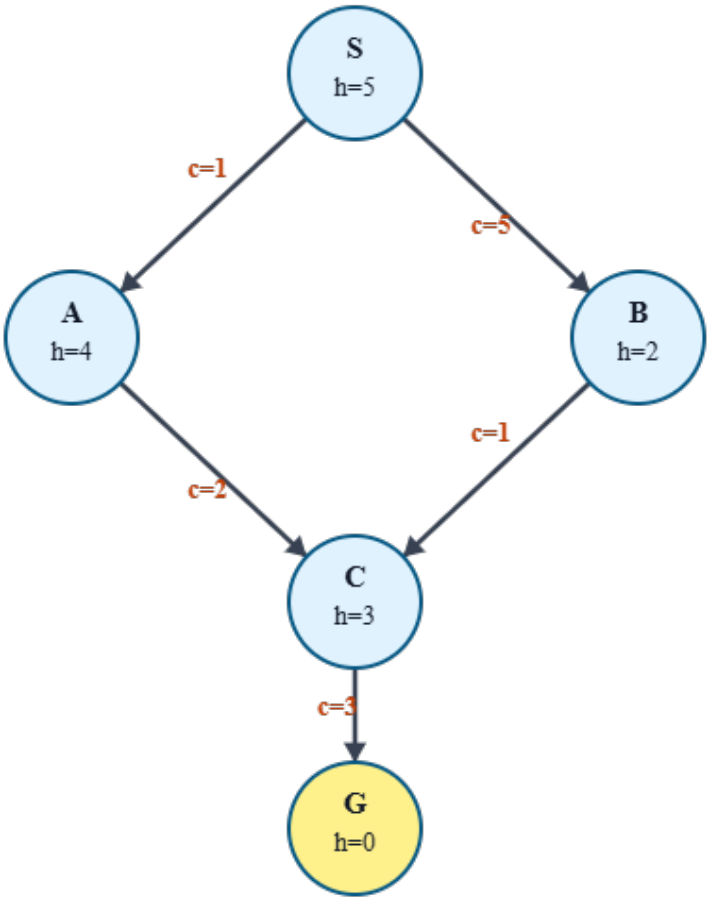
- Για τον **A**: $g(A) = g(S) + c(S, A) = 0 + 1 = 1$. $f(A) = g(A) + h(A) = 1 + 4 = 5$.
- Για τον **B**: $g(B) = g(S) + c(S, B) = 0 + 5 = 5$. $f(B) = g(B) + h(B) = 5 + 2 = 7$.
- **OPEN**: { A(f=5), B(f=7) }
- **CLOSED**: { S }

3. **Επέκταση του A**: Ο κόμβος με το μικρότερο f στην OPEN είναι ο A. Τον εξάγουμε. Εξετάζουμε τον γείτονά του, C.
   - Για τον **C**: $g(C) = g(A) + c(A, C) = 1 + 2 = 3$. $f(C) = g(C) + h(C) = 3 + 3 = 6$.
   - **OPEN**: { C(f=6), B(f=7) }
   - **CLOSED**: { S, A }
4. **Επέκταση του C**: Ο κόμβος με το μικρότερο f στην OPEN είναι ο C. Τον εξάγουμε. Εξετάζουμε τον γείτονά του, G.
   - Για τον **G**: $g(G) = g(C) + c(C, G) = 3 + 3 = 6$. $f(G) = g(G) + h(G) = 6 + 0 = 6$.
   - **OPEN**: { G(f=6), B(f=7) }
   - **CLOSED**: { S, A, C }
5. **Εύρεση Στόχου**: Ο κόμβος με το μικρότερο f στην OPEN είναι ο G. Ο G είναι η κατάσταση-στόχος. Ο αλγόριθμος τερματίζει επιτυχώς. Ανακατασκευάζοντας το μονοπάτι από τους γονείς (G←C←A←S), βρίσκουμε τη βέλτιστη λύση: **S** → **A** → **C** → **G** με συνολικό κόστος **6**.

Παρατηρούμε ότι παρόλο που ο κόμβος B φαινόταν αρχικά πιο ελκυστικός λόγω της χαμηλής ευρετικής του ($h(B) = 2$), ο αλγόριθμος τον αγνόησε σωστά, καθώς το υψηλό κόστος για να φτάσουμε σε αυτόν ($g(B) = 5$) αύξησε το συνολικό του f. Αυτό ακριβώς αναδεικνύει τη δύναμη του A* να εξισορροπεί το παρελθοντικό κόστος με το εκτιμώμενο μελλοντικό.

Πίνακας 2.1: Βήματα εκτέλεσης του A* στο παράδειγμα του Σχήματος 2.4.

Βήμα	**Επεξεργασία Κόμβου**	**OPEN Λίστα**	**CLOSED Λίστα**
1	–	{ S(5) }	{ }
2	S	{ A(5), B(7) }	{ S }
3	A	{ C(6), B(7) }	{ S, A }
4	C	{ G(6), B(7) }	{ S, A, C }
5	G	**Εύρεση Στόχου!**	{ S, A, C, G }



Σχήμα 2.4: Γράφος για το παράδειγμα εκτέλεσης του A*.

2.1.5 Ιδιότητες Ευρετικών Συναρτήσεων

Η απόδοση και, κυρίως, η ορθότητα του αλγορίθμου A* εξαρτώνται άμεσα από τις ιδιότητες της ευρετικής συνάρτησης $h(n)$ που χρησιμοποιείται. Η επιλογή της δεν είναι αυθαίρετη· για να εξασφαλιστούν συγκεκριμένες συμπεριφορές από τον αλγόριθμο, η $h(n)$ πρέπει να πληροί ορισμένες συνθήκες. Δύο τέτοιες ιδιότητες είναι θεμελιώδους σημασίας: η **παραδεκτότητα** και η **συνέπεια**.

Μια ευρετική συνάρτηση $h(n)$ ονομάζεται **παραδεκτή (admissible)** εάν για κάθε κόμβο n, η τιμή της δεν υπερεκτιμά ποτέ το πραγματικό κόστος για την επίτευξη του στόχου. Αν $h'(n)$ είναι το πραγματικό κόστος του βέλτιστου μονοπατιού από τον n στον στόχο, τότε μια ευρετική είναι παραδεκτή αν ισχύει:

$$h(n) \leq h'(n)$$

Μια παραδεκτή ευρετική είναι, κατά μία έννοια, "αισιόδοξη", καθώς η εκτίμηση που παρέχει είναι πάντα μικρότερη ή ίση με το πραγματικό κόστος. Η

ιδιότητα της παραδεκτότητας είναι η αναγκαία και ικανή συνθήκη για να εγγυηθεί ο αλγόριθμος A* την εύρεση της βέλτιστης (ελάχιστου κόστους) λύσης. Αν η ευρετική δεν ήταν παραδεκτή, θα μπορούσε να παραπλανήσει τον αλγόριθμο, κάνοντάς τον να θεωρήσει ένα υποβέλτιστο μονοπάτι ως πιο ακριβό από ό,τι είναι στην πραγματικότητα, και να καταλήξει σε μια μη βέλτιστη λύση. **Η διασφάλιση της παραδεκτότητας της ευρετικής συνάρτησης που θα αναπτυχθεί για τον εξειδικευμένο planner είναι υψίστης σημασίας, καθώς αποτελεί την προϋπόθεση για μια δίκαιη σύγκριση με τους βέλτιστους (optimal) domain-independent planners.**

Μια ισχυρότερη συνθήκη από την παραδεκτότητα είναι η **συνέπεια (consistency)**, γνωστή και ως **μονοτονία (monotonicity)**. Μια ευρετική είναι συνεπής εάν, για κάθε κόμβο n και κάθε γείτονά του n' που προκύπτει από την εφαρμογή της δράσης a, το εκτιμώμενο κόστος από το n δεν είναι ποτέ μεγαλύτερο από το κόστος της μετάβασης στο n' συν το εκτιμώμενο κόστος από το n'. Η συνθήκη αυτή, γνωστή και ως τριγωνική ανισότητα, εκφράζεται ως εξής (Σχήμα 2.4):

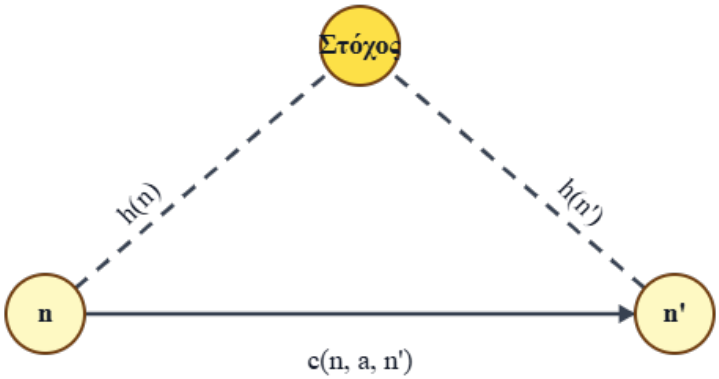
$$h(n) \leq c(n, a, n') + h'(n)$$

όπου $c(n, a, n')$ είναι το κόστος της δράσης a για τη μετάβαση από τον n στον n'. Κάθε συνεπής ευρετική είναι ταυτόχρονα και παραδεκτή (υπό την προϋπόθεση ότι $h(goal) = 0$). Η σημασία της συνέπειας είναι κυρίως πρακτική: όταν χρησιμοποιείται μια συνεπής ευρετική, ο αλγόριθμος A* είναι εγγυημένα πιο αποδοτικός. Αυτό συμβαίνει διότι, με μια συνεπή ευρετική, κάθε φορά που ο A* επιλέγει έναν κόμβο για επέκταση, έχει ήδη βρει το βέλτιστο μονοπάτι προς αυτόν. Κατά συνέπεια, κανένας κόμβος δεν χρειάζεται να επεξεργαστεί ξανά αφού μπει στη λίστα CLOSED, μειώνοντας τις συνολικές υπολογιστικές απαιτήσεις.

Για την παρούσα εργασία, η κατανόηση αυτών των ιδιοτήτων είναι κεντρικής σημασίας. Κατά τον σχεδιασμό των εξειδικευμένων ευρετικών συναρτήσεων για το πρόβλημα 'Rover', θα εξεταστεί όχι μόνο η απόδοσή τους, αλλά και το κατά πόσον πληρούν τις συνθήκες της παραδεκτότητας και της συνέπειας, καθώς αυτό επηρεάζει άμεσα τόσο την ποιότητα της λύσης όσο και την ταχύτητα εύρεσής της[1, 9].

2.2 Βασικές Αρχές Automated Planning

Αφού εξετάστηκαν οι θεμελιώδεις αλγόριθμοι αναζήτησης, το επόμενο βήμα είναι η εφαρμογή τους στο πεδίο του Αυτοματοποιημένου Σχεδιασμού. Στον σχεδιασμό, το πρόβλημα αναζήτησης αποκτά μια πιο δομημένη μορφή. Οι "καταστάσεις" δεν είναι απλοί κόμβοι σε έναν γράφο, αλλά λογικές περιγραφές του κόσμου, οι οποίες απαρτίζονται από ένα σύνολο κατηγορημάτων (predicates). Αντίστοιχα, οι "δράσεις" είναι τελεστές με συγκεκριμένες προϋποθέσεις (preconditions) και αποτελέσματα (effects) που μεταβάλλουν αυτές τις καταστάσεις. Για την τυποποιημένη περιγραφή αυτών των σύνθετων προβλημάτων, αναπτύχθηκαν εξει-



Σχήμα 2.5: Η συνθήκη της συνέπειας (Τριγωνική Ανισότητα).

δικευμένες γλώσσες, με κυρίαρχη τη PDDL. Η παρούσα ενότητα θα παρουσιάσει αυτή τη γλώσσα και στη συνέχεια θα εξετάσει τις δύο κύριες στρατηγικές με τις οποίες η αναζήτηση σε χώρους καταστάσεων εφαρμόζεται για την επίλυση προβλημάτων σχεδιασμού: τον σχεδιασμό προόδου (progression) και τον σχεδιασμό οπισθοδρόμησης (regression).

2.2.1 Η Γλώσσα PDDL

Για να μπορούν οι planners, που είναι συστήματα γενικής χρήσης, να επιλύουν οποιοδήποτε πρόβλημα, απαιτείται μια κοινή, τυποποιημένη γλώσσα για την περιγραφή του. Αυτή τη λειτουργία επιτελεί η **Γλώσσα Ορισμού Πεδίων Σχεδίασης (Planning Domain Definition Language - PDDL)**, η οποία αποτελεί το de facto πρότυπο στην κοινότητα του αυτοματοποιημένου σχεδιασμού. Ένα πρόβλημα σε PDDL ορίζεται από δύο αρχεία: το αρχείο του **πεδίου (domain)**, που περιγράφει τους γενικούς κανόνες και τις δυνατές δράσεις, και το αρχείο του **προβλήματος (problem)**, που ορίζει μια συγκεκριμένη περίπτωση προς επίλυση.

Το αρχείο του domain ορίζει τα στοιχεία που παραμένουν σταθερά σε όλες τις εκδοχές ενός προβλήματος. Περιλαμβάνει τον ορισμό των **κατηγορημάτων (predicates)**, τα οποία είναι λογικές προτάσεις που περιγράφουν τις ιδιότητες του κόσμου και τις σχέσεις μεταξύ των αντικειμένων (π.χ., $(in ?x - rover ?y - waypoint)$ στο πεδίο 'Rover', που δηλώνει ότι το όχημα $?x$ βρίσκεται στην τοποθεσία $?y$). Το πιο σημαντικό στοιχείο είναι οι **δράσεις (actions)**, οι οποίες περιγράφουν πώς μπορεί να αλλάξει η κατάσταση του κόσμου. Κάθε δράση αποτελείται από:

- Τις **παραμέτρους (parameters)** της δράσης.

- Τις **προϋποθέσεις (preconditions)** που πρέπει να ισχύουν για να μπορέσει να εκτελεστεί.
- Τα **αποτελέσματα (effects)**, δηλαδή τις αλλαγές που επιφέρει η δράση. Τα αποτελέσματα χωρίζονται σε λίστα προσθήκης (add list) (κατηγορήματα που γίνονται αληθή) και λίστα διαγραφής (delete list) (κατηγορήματα που γίνονται ψευδή).

Το αρχείο του προβλήματος, από την άλλη πλευρά, ορίζει τα συγκεκριμένα **αντικείμενα (objects)** που υπάρχουν σε μια περίπτωση, την **αρχική κατάσταση** (: $init$), δηλαδή το σύνολο των κατηγορημάτων που είναι αληθή στην αρχή, και τον **στόχο** (: $goal$), δηλαδή το σύνολο των κατηγορημάτων που πρέπει να ισχύουν στο τέλος [4]. Για παράδειγμα, η δράση $drop$ από το πεδίο 'Rover' παρουσιάζεται στο παρακάτω παράδειγμα.

```
(:action drop

   :parameters (?x - rover ?y - store)     ; Η δράση αφορά ένα όχημα ?x και τον
                                           αποθηκευτικό του χώρο ?y

   :precondition (and (store_of ?y ?x)     ; Προϋπόθεση 1: Το ?y πρέπει να είναι ο
                                           χώρος του ?x
                      (full ?y))           ; Προϋπόθεση 2: Ο χώρος ?y πρέπει να
                                           είναι γεμάτος

   :effect (and (not (full ?y)))           ; Αποτέλεσμα 1 (delete list): Ο χώρος ?y
                                           δεν είναι πλέον γεμάτος
               (empty ?y))                 ; Αποτέλεσμα 2 (add list): Ο χώρος ?y εί-
                                           ναι πλέον άδειος

)
```

Σχήμα 2.6: Παράδειγμα μιας δράσης σε PDDL: drop

2.2.2 Σχεδιασμός σε Χώρο Καταστάσεων (State-Space Planning)

Έχοντας ορίσει τη γλώσσα PDDL, μπορούμε πλέον να δούμε πώς το πρόβλημα του αυτοματοποιημένου σχεδιασμού ανάγεται σε ένα πρόβλημα αναζήτησης σε χώρους καταστάσεων. Σε αυτό το πλαίσιο, κάθε κόμβος του γράφου αναζήτησης αντιστοιχεί σε μια πλήρη κατάσταση του κόσμου, η οποία περιγράφεται από το σύνολο των κατηγορημάτων που είναι αληθή εκείνη τη στιγμή. Η αναζήτηση για ένα πλάνο μετατρέπεται, συνεπώς, σε μια αναζήτηση για ένα μονοπάτι

από την αρχική κατάσταση (: $init$) στην κατάσταση-στόχο (: $goal$). Υπάρχουν δύο κυρίαρχες στρατηγικές για τη διερεύνηση αυτού του χώρου: η αναζήτηση προς τα εμπρός (progression) και η αναζήτηση προς τα πίσω (regression).

Ο **Σχεδιασμός Προόδου (Progression)**, γνωστός και ως αναζήτηση προς τα εμπρός (forward state-space search), είναι η πιο ευθύς προσέγγιση. Η διαδικασία ξεκινά από την αρχική κατάσταση, όπως αυτή ορίζεται στο αρχείο του προβλήματος. Σε κάθε βήμα, ο αλγόριθμος εξετάζει όλες τις δράσεις των οποίων οι προϋποθέσεις ικανοποιούνται από την τρέχουσα κατάσταση. Για κάθε τέτοια εφαρμόσιμη δράση, δημιουργείται ένας κόμβος-διάδοχος (successor node) εφαρμόζοντας τα αποτελέσματα (effects) της δράσης στην τρέχουσα κατάσταση. Το κύριο πλεονέκτημα αυτής της μεθόδου είναι ότι επιτρέπει τη χρήση πολύ ισχυρών και πληροφοριακών ευρετικών συναρτήσεων, καθώς σε κάθε σημείο είναι γνωστή η πλήρης, τρέχουσα κατάσταση του κόσμου. Ωστόσο, το βασικό της μειονέκτημα είναι ο δυνητικά τεράστιος **παράγοντας διακλάδωσης (branching factor)**. Σε προβλήματα με πολλά αντικείμενα και δράσεις, ο αριθμός των εφαρμόσιμων δράσεων σε κάθε κατάσταση μπορεί να είναι πολύ μεγάλος, οδηγώντας σε μια εκρηκτική αύξηση του χώρου αναζήτησης [2].

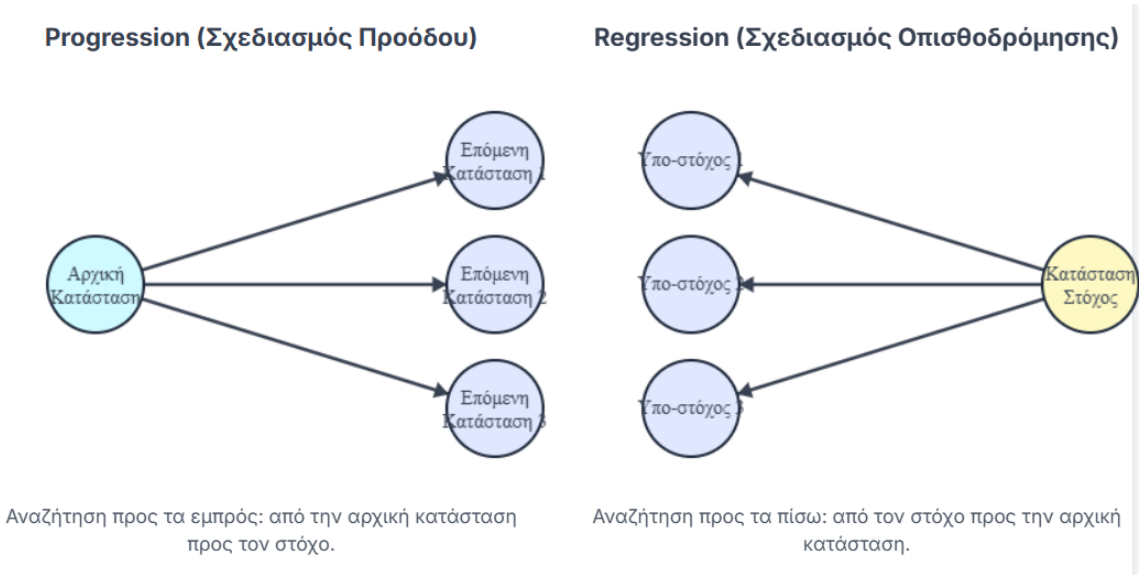
Αντίθετα, ο **Σχεδιασμός Οπισθοδρόμησης (Regression)**, ή αναζήτηση προς τα πίσω (backward search), λειτουργεί αντίστροφα. Ξεκινά από την περιγραφή του στόχου (: $goal$) και εφαρμόζει τις δράσεις "προς τα πίσω". Σε κάθε βήμα, ο αλγόριθμος αναζητά μια δράση που θα μπορούσε να επιτύχει ένα από τα τρέχοντα υπο-στοχαστικά κατηγορήματα. Ο κόμβος-προκάτοχος (predecessor node) δημιουργείται αντικαθιστώντας το κατηγόρημα-αποτέλεσμα με τις προϋποθέσεις της δράσης αυτής [1]. Το πλεονέκτημα της regression είναι ότι εξετάζει μόνο τις δράσεις που είναι σχετικές με την επίτευξη του στόχου, αποφεύγοντας πολλές άσκοπες επεκτάσεις. Αυτό μπορεί να οδηγήσει σε σημαντικά μικρότερους χώρους αναζήτησης. Το μειονέκτημά της, ωστόσο, είναι η δυσκολία στον χειρισμό σύνθετων στόχων και στη δημιουργία εξίσου ισχυρών ευρετικών συναρτήσεων με αυτές της progression [8].

2.3 Προηγμένες μέθοδοι Planning

Πέρα από τις θεμελιώδεις μεθόδους αναζήτησης σε χώρο καταστάσεων που εξετάστηκαν προηγουμένως, η έρευνα στον αυτοματοποιημένο σχεδιασμό έχει αναπτύξει και άλλες, πιο προηγμένες τεχνικές για την αντιμετώπιση της πολυπλοκότητας των προβλημάτων. Η παρούσα ενότητα εξετάζει δύο τέτοιες σημαντικές ιδέες που επηρέασαν την εξέλιξη του πεδίου. Αρχικά, θα παρουσιαστεί ο **Σχεδιασμός Μερικής Διάταξης (Partial-Order Planning)**, μια εναλλακτική προσέγγιση που δεν κατασκευάζει ένα γραμμικό πλάνο, αλλά ένα πιο ευέλικτο σύνολο δράσεων με περιορισμούς διάταξης. Στη συνέχεια, θα εστιάσουμε στον τρόπο με τον οποίο μπορούν να εξαχθούν αυτόματα ισχυρές **ευρετικές συναρτήσεις** απευθείας από την περιγραφή ενός προβλήματος, με έμφαση στη

Πίνακας 2.2: Συγκριτική Ανάλυση του Σχεδιασμού Progression και Regression.

Χαρακτηριστικό	**Σχεδιασμός Προόδου (Progression)**	**Σχεδιασμός Οπισθοδρόμησης (Regression)**
Κατεύθυνση Αναζήτησης	Από την **αρχική κατάσταση** προς τον στόχο.	Από την **κατάσταση-στόχο** προς την αρχική.
Αναπαράσταση Κόμβου	Μια **πλήρης, συγκεκριμένη κατάσταση** του κόσμου (σύνολο αληθών κατηγορημάτων).	Ένας **υπο-στόχος** (σύνολο κατηγορημάτων που πρέπει να ισχύουν).
Παράγοντας Διακλάδωσης	Συνήθως **πολύ μεγάλος** (όλες οι εφαρμόσιμες δράσεις).	Δυνητικά **μικρότερος** (μόνο οι δράσεις που είναι σχετικές με τον στόχο).
Ισχύς Ευρετικών	Επιτρέπει τον υπολογισμό **πολύ ισχυρών ευρετικών** (π.χ., delete-relaxation).	Ο υπολογισμός ισχυρών ευρετικών είναι **πιο δύσκολος**.



Σχήμα 2.7: Σύγκριση Σχεδιασμού Progression και Regression.

θεμελιώδη τεχνική της **χαλάρωσης διαγραφών (delete-relaxation)**, η οποία αποτελεί τη βάση για τους περισσότερους σύγχρονους planners.

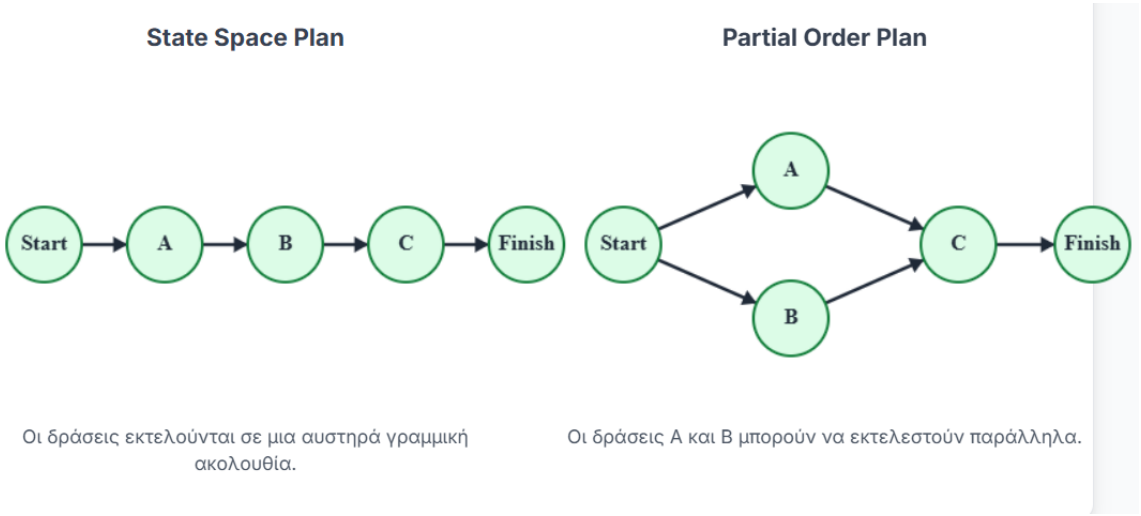
2.3.1 Σχεδιασμός Μερικής Διάταξης (Partial-Order Planning)

Σε αντίθεση με τις μεθόδους σχεδιασμού σε χώρο καταστάσεων (progression / regression) που κατασκευάζουν μια πλήρως διατεταγμένη ακολουθία δράσεων, ο **Σχεδιασμός Μερικής Διάταξης (Partial-Order Planning - POP)** ακολουθεί μια πιο ευέλικτη στρατηγική. Βασίζεται στην αρχή της **ελάχιστης δέσμευσης (least commitment)**: ο αλγόριθμος καθυστερεί τη λήψη αποφάσεων για τη σειρά των δράσεων για όσο το δυνατόν περισσότερο, προσθέτοντας έναν περιορισμό διάταξης μόνο όταν είναι απολύτως απαραίτητο για την επίλυση μιας σύγκρουσης. Το αποτέλεσμα δεν είναι ένα γραμμικό μονοπάτι, αλλά ένα πλάνο που αποτελείται από ένα σύνολο δράσεων και ένα σύνολο από περιορισμούς διάταξης (π.χ., η δράση Α πρέπει να εκτελεστεί πριν τη δράση Β) [2].

Η αναζήτηση στον POP δεν γίνεται στον χώρο των καταστάσεων του κόσμου, αλλά στον **χώρο των (ατελών) πλάνων**. Ο αλγόριθμος ξεκινά με ένα κενό πλάνο που περιέχει μόνο δύο ψευδο-δράσεις: τη Start, της οποίας τα effects είναι η αρχική κατάσταση, και τη Finish, της οποίας οι preconditions είναι οι συνθήκες-στόχοι. Στη συνέχεια, ο planner αναζητά "ατέλειες" (flaws) στο τρέχον πλάνο και προσπαθεί να τις διορθώσει. Μια τυπική ατέλεια είναι μια "**ανοικτή προϋπόθεση**" **(open precondition)**, δηλαδή μια προϋπόθεση μιας δράσης που δεν έχει ακόμη ικανοποιηθεί. Για να διορθώσει μια τέτοια ατέλεια, ο planner μπορεί να προσθέσει στο πλάνο μια νέα δράση (ή να χρησιμοποιήσει μια υπάρχουσα) που πετυχαίνει την προϋπόθεση αυτή, δημιουργώντας έναν **αιτιατό σύνδεσμο (causal link)** μεταξύ των δύο δράσεων [1].

Το κύριο πλεονέκτημα αυτής της προσέγγισης είναι η δυνατότητα αποσύνθεσης του προβλήματος σε μικρότερα, ανεξάρτητα υποπροβλήματα, τα οποία μπορούν να επιλυθούν παράλληλα. Ωστόσο, οι planners μερικής διάταξης αποδείχθηκαν στην πράξη λιγότερο αποδοτικοί από τους σύγχρονους planners που βασίζονται σε ευρετική αναζήτηση στον χώρο καταστάσεων, κυρίως λόγω της δυσκολίας στον ορισμό ισχυρών ευρετικών συναρτήσεων για την καθοδήγηση της αναζήτησης στον χώρο των πλάνων.

Η παρουσίαση αυτής της εναλλακτικής προσέγγισης, παρότι δεν αποτελεί τον πυρήνα της παρούσας εργασίας, είναι σημαντική. Λειτουργεί ως σημείο αναφοράς που, μέσω της αντίθεσης, αναδεικνύει την κυριαρχία και την πρακτική αποτελεσματικότητα της ευρετικής αναζήτησης σε χώρο καταστάσεων, της μεθοδολογίας δηλαδή που βρίσκεται στο επίκεντρο της υλοποίησης και της ανάλυσης που ακολουθεί.



Σχήμα 2.8: Σύγκριση State-Space και Partial-Order Planning.

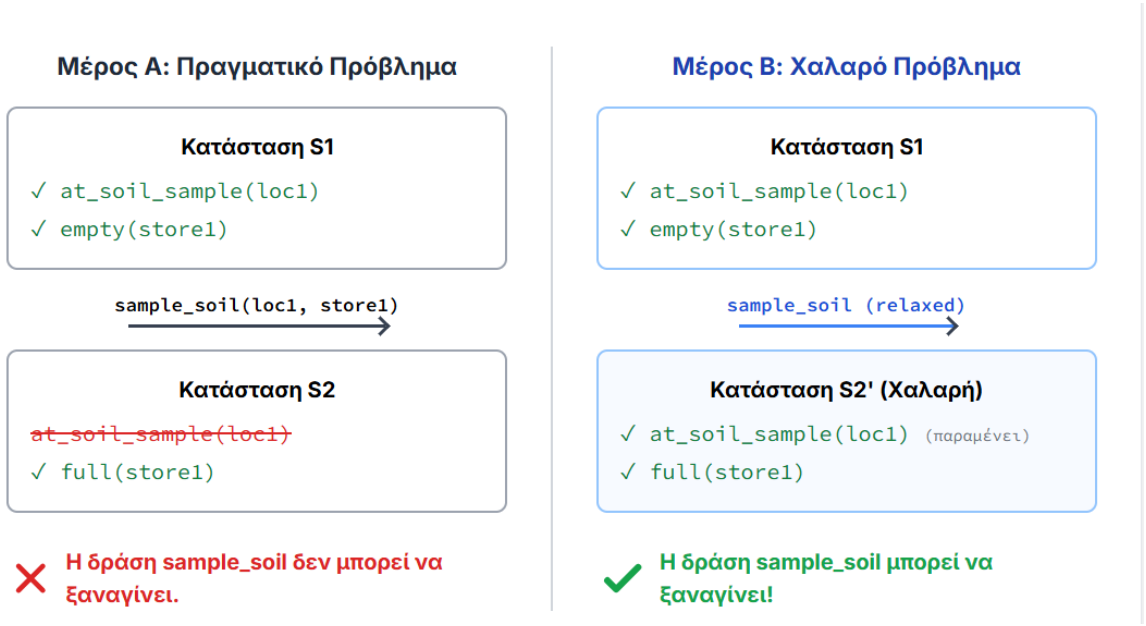
2.3.2 Ευρετικές Συναρτήσεις για Προβλήματα Planning

Η αποτελεσματικότητα των planners που βασίζονται σε ευρετική αναζήτηση, όπως ο A*, εξαρτάται απόλυτα από την ποιότητα της ευρετικής συνάρτησης h(n). Σε αντίθεση με προβλήματα όπου η ευρετική μπορεί να οριστεί εύκολα (π.χ., η ευθεία απόσταση σε ένα πρόβλημα πλοήγησης), στον αυτοματοποιημένο σχεδιασμό προκύπτει το ερώτημα: πώς μπορεί ένας domain-independent planner να υπολογίσει μια καλή ευρετική, αντλώντας πληροφορία απευθείας από την περιγραφή PDDL του προβλήματος; Η κυρίαρχη προσέγγιση είναι η **επίλυση ενός απλοποιημένου (relaxed) προβλήματος**. Η ιδέα είναι να μετασχηματιστεί το αρχικό, δύσκολο πρόβλημα σε ένα ευκολότερο, του οποίου η λύση μπορεί να βρεθεί γρήγορα. Το κόστος της λύσης αυτού του χαλαρού προβλήματος χρησιμοποιείται στη συνέχεια ως η ευρετική εκτίμηση για το αρχικό πρόβλημα [4].

Η πιο διαδεδομένη και επιδραστική τεχνική για τη δημιουργία τέτοιων ευρετικών είναι η **χαλάρωση διαγραφών (delete-relaxation)**. Η κεντρική ιδέα είναι η εξής: κατασκευάζουμε ένα απλοποιημένο πρόβλημα αγνοώντας όλα τα αρνητικά αποτελέσματα (delete lists) των δράσεων. Σε αυτό το "χαλαρό" πρόβλημα, τα κατηγορήματα που γίνονται αληθή δεν μπορούν ποτέ να γίνουν ξανά ψευδή. **Για παράδειγμα, στο πεδίο 'Rover', η δράση** $sample\_soil$ **έχει ως αποτέλεσμα τη διαγραφή του κατηγορήματος (**$at\_soil\_sample?w$**), εμποδίζοντας τη λήψη πολλαπλών δειγμάτων από την ίδια τοποθεσία. Σε ένα πρόβλημα με χαλάρωση διαγραφών, αυτό το αρνητικό αποτέλεσμα αγνοείται, με αποτέλεσμα ο planner να θεωρεί λανθασμένα ότι μπορεί να συλλέξει άπειρα δείγματα από το ίδιο σημείο.** Κάθε δράση, λοιπόν, απλώς προσθέτει νέα αληθή κατηγορήματα, κάνοντας το πρόβλημα μονοτονικό και πολύ πιο εύκολο να επιλυθεί. Το κόστος του βέλτιστου πλάνου σε αυτό το χαλαρό πρόβλημα (το relaxed plan)

αποτελεί μια παραδεκτή και πολύ πληροφοριακή ευρετική για το αρχικό, μη-χαλαρό πρόβλημα. Στο Σχήμα 2.8 μπορείτε να δείτε αυτό το παράδειγμα και οπτικοποιημένο.

Η πιο γνωστή υλοποίηση αυτής της ιδέας είναι η ευρετική του planner **FF (Fast Forward)**, η οποία υπολογίζει το χαλαρό πλάνο χρησιμοποιώντας μια δομή δεδομένων που ονομάζεται **Γράφος Χαλαρού Πλάνου (Relaxed Planning Graph - RPG)**. Ο γράφος αυτός κατασκευάζεται σε επίπεδα που εναλλάσσονται μεταξύ κατηγορημάτων και δράσεων, δείχνοντας ποιες δράσεις μπορούν να εκτελεστούν σε κάθε "βήμα" και ποια νέα κατηγορήματα γίνονται αληθή. Η ανάλυση αυτού του γράφου επιτρέπει στον planner να εκτιμήσει γρήγορα πόσες δράσεις απαιτούνται για την επίτευξη των στόχων, παρέχοντας έτσι την τιμή της ευρετικής $h(n)$ [10]. Μια άλλη, εξίσου ισχυρή προσέγγιση, είναι η χρήση **ορόσημων (landmarks)**, τα οποία είναι κατηγορήματα που πρέπει υποχρεωτικά να ισχύσουν σε κάποιο σημείο κάθε έγκυρου πλάνου. Η ποιότητα των ευρετικών που βασίζονται σε ορόσημα έχει αποδειχθεί στην πράξη, με planners όπως ο **LAMA** να επιτυγχάνουν κορυφαία απόδοση σε διαγωνισμούς σχεδιασμού [11].



Σχήμα 2.9: Οπτικοποίηση της Χαλάρωσης Διαγραφών.

2.4 Αριθμητικός Σχεδιασμός (Numeric Planning)

Ενώ οι προηγμένες μέθοδοι, όπως η χαλάρωση διαγραφών, αποδείχθηκαν εξαιρετικά ισχυρές για την επίλυση κλασικών προβλημάτων planning, ένα μεγάλο μέρος των ρεαλιστικών εφαρμογών παρουσιάζει μια επιπλέον, σημαντική

πρόκληση: την ανάγκη για διαχείριση αριθμητικών ποσοτήτων. Αυτή η απαίτηση οδήγησε στη διαμόρφωση του υποπεδίου του **Αριθμητικού Σχεδιασμού (Numeric Planning)**. Η παρούσα ενότητα εστιάζει σε αυτή την κατηγορία προβλημάτων. Αρχικά, θα αναλυθεί γιατί η εισαγωγή αριθμητικών μεταβλητών (numeric fluents) αυξάνει δραματικά την πολυπλοκότητα του σχεδιασμού. Στη συνέχεια, θα παρουσιαστούν οι θεμελιώδεις προσεγγίσεις που αναπτύχθηκαν για την αντιμετώπιση αυτών των προκλήσεων, εξετάζοντας πώς οι ιδέες από την κλασική σχεδίαση επεκτάθηκαν για να καλύψουν και τις αριθμητικές διαστάσεις ενός προβλήματος.

2.4.1 Η Πρόκληση των Αριθμητικών Μεταβλητών

Η θεμελιώδης διαφορά του αριθμητικού σχεδιασμού από τον κλασικό έγκειται στην εισαγωγή των **αριθμητικών χαρακτηριστικών (numeric fluents)**. Αυτά είναι μεταβλητές που μπορούν να λάβουν πραγματικές (real) ή ακέραιες (integer) τιμές, σε αντίθεση με τα λογικά κατηγορήματα που είναι είτε αληθή είτε ψευδή. Μέσω αυτών, μπορούμε να μοντελοποιήσουμε ποσότητες όπως η ενέργεια, η απόσταση, ο χρόνος, ή το κόστος. Οι δράσεις μπορούν πλέον να έχουν αριθμητικές προϋποθέσεις (π.χ. $(>= (energy ?r) 8)$) και αριθμητικά αποτελέσματα (π.χ. $(decrease (energy ?r) 8)$).

Η εισαγωγή αυτών των μεταβλητών, παρότι αυξάνει δραματικά την εκφραστικότητα της γλώσσας, έχει μια εκρηκτική συνέπεια στον χώρο αναζήτησης. Στην κλασική σχεδίαση, ο αριθμός των πιθανών καταστάσεων είναι πεπερασμένος, αν και πολύ μεγάλος, καθώς ορίζεται από όλους τους πιθανούς συνδυασμούς των αληθών/ψευδών τιμών των κατηγορημάτων. Αντιθέτως, όταν έστω και μία αριθμητική μεταβλητή μπορεί να πάρει άπειρες πραγματικές τιμές, ο χώρος καταστάσεων γίνεται **θεωρητικά άπειρος**. Ακόμα και αν οι τιμές είναι διακριτές, ο αριθμός των συνδυασμών είναι τόσο αστρονομικός που καθιστά οποιαδήποτε προσπάθεια πλήρους εξερεύνησης του χώρου, με μεθόδους όπως η BFS ή η DFS, υπολογιστικά ανέφικτη. Είναι αδύνατο να απαριθμήσουμε και να αποθηκεύσουμε όλες τις πιθανές καταστάσεις [12].

Αυτή η "κατάρα της διαστατικότητας" (curse of dimensionality) που επιφέρουν οι αριθμητικές μεταβλητές, καθιστά σαφές ότι απαιτούνται πιο έξυπνες τεχνικές από την απλή αναζήτηση για την επίλυση τέτοιων προβλημάτων.

2.4.2 Προσεγγίσεις στον Αριθμητικό Σχεδιασμό

Για την αντιμετώπιση του προβλήματος του άπειρου χώρου καταστάσεων, η έρευνα στον αριθμητικό σχεδιασμό στράφηκε στην επέκταση των επιτυχημένων ιδεών από την κλασική σχεδίαση. Η κυρίαρχη στρατηγική παρέμεινε η εξαγωγή ευρετικών συναρτήσεων μέσω της επίλυσης ενός απλοποιημένου, **χαλαρού προβλήματος**.

Μια από τις πρώτες και πιο επιδραστικές προσεγγίσεις για την αντιμετώπιση του αριθμητικού σχεδιασμού ήταν η επέκταση της ιδέας της χαλάρωσης διαγραφών. Η ιδέα αυτή, που αποτέλεσε τη βάση για την επιτυχία του planner **FF** στην κλασική σχεδίαση [10], επεκτάθηκε αργότερα για τον χειρισμό αριθμητικών μεταβλητών στο σύστημα **Metric-FF** [12]. Αυτή η προσέγγιση παράγει μια πληροφοριακή αλλά μη-παραδεκτή ευρετική, κατάλληλη για ικανοποιητική (satisficing) αλλά όχι για βέλτιστη (optimal) σχεδίαση.

Η μετέπειτα έρευνα για τη βέλτιστη αριθμητική σχεδίαση επικεντρώθηκε στην ανάπτυξη παραδεκτών ευρετικών. Αντί να εφεύρουν ριζικά νέες ιδέες, οι πιο επιτυχημένες σύγχρονες προσεγγίσεις προσάρμοσαν ισχυρές ιδέες από την κλασική σχεδίαση. Για παράδειγμα, η τεχνική του **subgoaling** γενικεύτηκε για να χειρίζεται αριθμητικές συνθήκες [13], ενώ η έννοια των **οροσήμων (landmarks)** επεκτάθηκε για την αυτόματη εξαγωγή κρίσιμων αριθμητικών περιορισμών που πρέπει να ικανοποιηθούν κατά τη διάρκεια ενός πλάνου [14]. Επιπλέον, αναπτύχθηκαν εξειδικευμένες τεχνικές για τη διαχείριση **συνεχών πόρων (continuous resources)** και **χρονικών περιορισμών (temporal constraints)**, που είναι κρίσιμες για πεδία όπως το 'Rover' [15, 16]. Αυτές οι εξελίξεις κατέστησαν εφικτή την εύρεση βέλτιστων λύσεων σε απαιτητικά αριθμητικά προβλήματα.

Για τον λόγο αυτό, οι ευρετικές τύπου Metric-FF είναι κατάλληλες αποκλειστικά για **ικανοποιητική σχεδίαση (satisficing planning)**, όπου ο στόχος είναι η γρήγορη εύρεση μιας *οποιασδήποτε* έγκυρης λύσης, ανεξαρτήτως του κόστους της. Δεν μπορούν, ωστόσο, να εγγυηθούν τη βελτιστότητα, και συνεπώς δεν είναι κατάλληλες για **βέλτιστη σχεδίαση (optimal planning)**, όπου απαιτείται η λύση με το ελάχιστο δυνατό κόστος. **Αυτή ακριβώς η πρόκληση –η δημιουργία μιας παραδεκτής ευρετικής για ένα πολύπλοκο αριθμητικό πρόβλημα όπως το 'Rover'– αποτελεί το σημείο όπου ο εξειδικευμένος, domain-dependent planner μπορεί να προσφέρει σημαντικά πλεονεκτήματα, ένα ζήτημα που θα εξεταστεί εμπειρικά στο Κεφάλαιο 4.**

2.5 Η Αντιπαράθεση Γενικότητας και Εξειδίκευσης στον Αυτοματοποιημένο Σχεδιασμό

Ένα θεμελιώδες δίλημμα που διατρέχει την ιστορία του αυτοματοποιημένου σχεδιασμού είναι η αντιπαράθεση μεταξύ της γενικότητας και της εξειδίκευσης.

Από τη μία πλευρά, οι **planners ανεξάρτητοι από το πεδίο (domain-independent planners)** αντιπροσωπεύουν τον πυρήνα της ακαδημαϊκής έρευνας. Στόχος τους είναι η ανάπτυξη αλγορίθμων ικανών να επιλύσουν οποιοδήποτε πρόβλημα μπορεί να περιγραφεί σε μια τυποποιημένη γλώσσα όπως η PDDL. Το μεγάλο τους πλεονέκτημα είναι η ευελιξία και η καθολικότητά τους. Ωστόσο, ακριβώς επειδή αγνοούν τις ιδιαιτερότητες του κάθε προβλήματος, συχνά αποτυγχάνουν να εκμεταλλευτούν τη δομή του, οδηγώντας σε χαμηλότερη

Πίνακας 2.3: Σύνοψη Προκλήσεων και Προσεγγίσεων στον Αριθμητικό Σχεδιασμό.

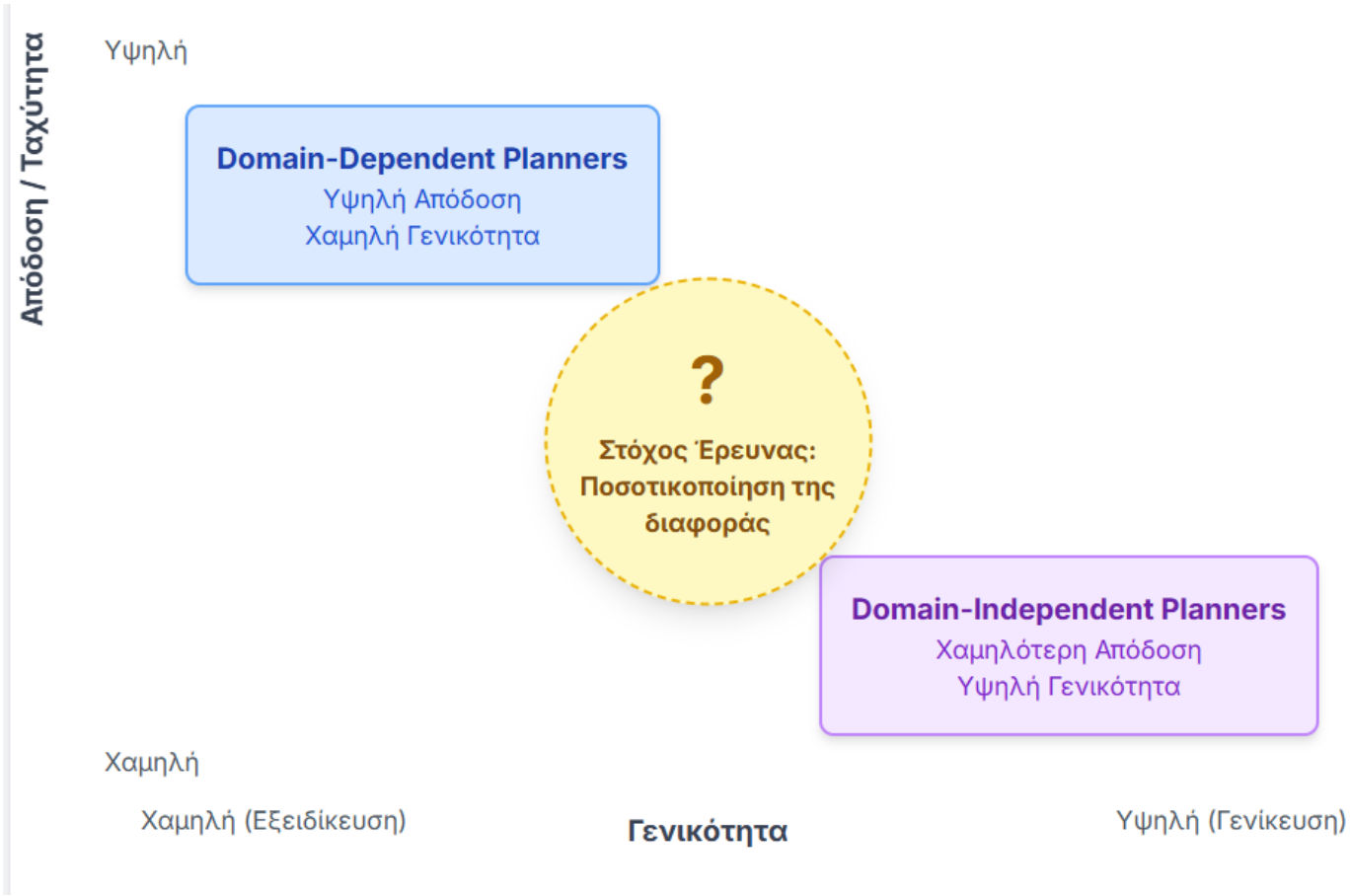
Η Πρόκληση (Το Πρόβλημα)	**Γιατί είναι Πρόβλημα;**	**Η Λύση / Προσέγγιση**
Άπειρος Χώρος Καταστάσεων	Οι κλασικοί αλγόριθμοι δεν μπορούν να εξερευνήσουν έναν άπειρο χώρο.	Δεν εξερευνούμε τον χώρο, αλλά τον "εκτιμούμε". Δημιουργούμε **ευρετικές συναρτήσεις** λύνοντας μια πιο απλή, "χαλαρή" εκδοχή του προβλήματος.
Εύρεση Βέλτιστης Λύσης	Οι πρώτες "χαλαρές" ευρετικές (όπως του Metric-FF) είναι γρήγορες αλλά **όχι παραδεκτές**, άρα δεν εγγυώνται τη βέλτιστη λύση.	Διάκριση των planners σε: α) **Ικανοποιητικούς (Satisficing)**, που βρίσκουν γρήγορα μια οποιαδήποτε λύση, και β) **Βέλτιστους (Optimal)**, που εγγυώνται την καλύτερη.

απόδοση [17]. Η έρευνα σε αυτόν τον τομέα εστιάζει στη δημιουργία ολοένα και πιο "έξυπνων" ευρετικών και τεχνικών αναζήτησης που μπορούν αυτόματα να ανακαλύψουν χρήσιμες ιδιότητες ενός προβλήματος [18].

Στον αντίποδα, οι **planners εξαρτώμενοι από το πεδίο (domain-dependent planners)** είναι συστήματα σχεδιασμένα "στο χέρι" για ένα και μόνο πεδίο προβλημάτων. Ενσωματώνοντας βαθιά, εξειδικευμένη γνώση, μπορούν να επιτύχουν εξαιρετικά υψηλή απόδοση. Το μειονέκτημά τους είναι το υψηλό κόστος ανάπτυξης και η πλήρης έλλειψη γενικότητας: ένας planner για το 'Rover' δεν μπορεί να λύσει ένα πρόβλημα logistics [19].

Η σύγχρονη έρευνα προσπαθεί να γεφυρώσει αυτό το χάσμα με υβριδικές μεθόδους, όπως η αυτόματη διαμόρφωση (automatic configuration) των παραμέτρων ενός planner [20] ή η επιλογή του καταλληλότερου planner από ένα χαρτοφυλάκιο (portfolio) [21], ανάλογα με τα χαρακτηριστικά του προβλήματος.

Αυτό ακριβώς το δίλημμα βρίσκεται στον πυρήνα της παρούσας εργασίας. Σκοπός μας είναι να ποσοτικοποιήσουμε εμπειρικά αυτόν τον συμβιβασμό, αναπτύσσοντας έναν εξειδικευμένο planner για ένα απαιτητικό αριθμητικό πρόβλημα και συγκρίνοντάς τον με τα πιο σύγχρονα συστήματα γενικού σκοπού, όπως αυτά που διακρίθηκαν στον πρόσφατο διαγωνισμό IPC 2023.



Σχήμα 2.10: Ο Συμβιβασμός Γενικότητας και Απόδοσης.

Κεφάλαιο 3

Το πρόβλημα

3.1 Ο Διεθνής Διαγωνισμός Σχεδιασμού (International Planning Competition - IPC)

Σε έναν εφαρμοσμένο τομέα της επιστήμης των υπολογιστών, όπως ο αυτοματοποιημένος σχεδιασμός, η πρόοδος δεν μετριέται μόνο με θεωρητικές αποδείξεις, αλλά κυρίως μέσα από την εμπειρική αξιολόγηση και τη συγκριτική ανάλυση των συστημάτων που αναπτύσσονται. Το κατεξοχήν πλαίσιο για αυτή την αξιολόγηση είναι ο **Διεθνής Διαγωνισμός Σχεδιασμού (International Planning Competition - IPC)**. Ο IPC, που διοργανώνεται περιοδικά στο πλαίσιο του κορυφαίου συνεδρίου ICAPS (International Conference on Automated Planning and Scheduling), λειτουργεί ως το de facto πρότυπο (benchmark) της ακαδημαϊκής κοινότητας. Παρέχει ένα κοινό σύνολο προβλημάτων-αναφοράς (benchmark problems) και μια αυστηρή μεθοδολογία αξιολόγησης, επιτρέποντας την αντικειμενική σύγκριση της απόδοσης διαφορετικών planners σε όρους ταχύτητας, ποιότητας των παραγόμενων πλάνων και ικανότητας επίλυσης δύσκολων προβλημάτων. Η παρούσα ενότητα θα παρουσιάσει την ιστορία και τη σημασία του διαγωνισμού, θέτοντας το πλαίσιο μέσα στο οποίο θα αξιολογηθεί η συνεισφορά της παρούσας εργασίας.

3.1.1 Ιστορία και Σημασία

Ο Διεθνής Διαγωνισμός Σχεδιασμού ξεκίνησε το 1998 με σκοπό να δώσει λύση σε ένα χρόνιο πρόβλημα της ερευνητικής κοινότητας: την έλλειψη μιας κοινής, αντικειμενικής μεθόδου για τη μέτρηση της προόδου και τη σύγκριση των διαφορετικών συστημάτων σχεδιασμού. Πριν τον IPC, οι ερευνητές συχνά αξιολογούσαν τους planners τους σε ιδιωτικά, μη-τυποποιημένα σύνολα προβλημάτων, καθιστώντας τη σύγκριση μεταξύ διαφορετικών συστημάτων δύσκολη, αν όχι αδύνατη. Ο διαγωνισμός καθιέρωσε ένα κοινό πλαίσιο αξιολόγησης, παρέχοντας μια σουίτα από προβλήματα-αναφοράς (benchmarks) και σαφή κριτήρια

μέτρησης της απόδοσης, μετατρέποντας την εμπειρική αξιολόγηση σε κεντρικό πυλώνα της έρευνας [3].

Η σημασία του IPC, ωστόσο, εκτείνεται πέρα από την απλή κατάταξη των planners. Λειτουργεί ως ένας ισχυρός καταλύτης για την πρόοδο, καθώς κάθε νέος διαγωνισμός εισάγει νέα, πιο απαιτητικά προβλήματα και κατηγορίες (tracks), ωθώντας την κοινότητα να αναπτύξει καινοτόμες τεχνικές. Πολλές από τις πιο επιδραστικές ιδέες στον σύγχρονο σχεδιασμό, όπως οι ευρετικές που βασίζονται σε γράφους χαλαρού πλάνου (RPG) ή σε ορόσημα (landmarks), αναδείχθηκαν και επικυρώθηκαν μέσα από την επιτυχία τους στον IPC. Με αυτόν τον τρόπο, ο διαγωνισμός όχι μόνο μετρά την πρόοδο, αλλά την καθοδηγεί ενεργά, επιβραβεύοντας τις προσεγγίσεις που αποδεικνύονται πιο αποτελεσματικές στην πράξη [5]. Συνεπώς, η απόδοση ενός planner στον IPC θεωρείται σήμερα η πιο αξιόπιστη ένδειξη της κατάστασης της τέχνης (state-of-the-art) στον τομέα.

3.1.2 Εξέλιξη και στόχοι

Στις πρώτες του εκδόσεις, ο διαγωνισμός IPC εστίαζε αποκλειστικά στην κλασική σχεδίαση, όπου οι planners καλούνταν να λύσουν προβλήματα με ντετερμινιστικές δράσεις και λογικές συνθήκες. Αυτή η προσέγγιση, αν και θεμελιώδης, απείχε σημαντικά από την πολυπλοκότητα των πραγματικών εφαρμογών.

Το σημείο καμπής στην εξέλιξη του διαγωνισμού ήρθε με την εισαγωγή της γλώσσας PDDL2.1 το 2002 (αναλύθηκε στο Κεφάλαιο 2). Αυτή η νέα έκδοση της γλώσσας παρείχε για πρώτη φορά έναν τυποποιημένο τρόπο για την περιγραφή προβλημάτων που περιελάμβαναν **χρόνο (durative actions)** και **αριθμητικές μεταβλητές (numeric fluents)**, όπως πόρους, αποστάσεις ή κόστος [4]. Η δυνατότητα αυτή άνοιξε τον δρόμο για τη μοντελοποίηση πολύ πιο ρεαλιστικών και απαιτητικών σεναρίων, καθιστώντας όμως την απευθείας σύγκριση των κλασικών planners με τα νέα, πιο εξειδικευμένα συστήματα, προβληματική.

Για να αντιμετωπιστεί αυτή η πρόκληση, ο IPC υιοθέτησε σταδιακά τη δομή των **εξειδικευμένων κατηγοριών (specialized tracks)**. Ο στόχος ήταν η δημιουργία διακριτών πεδίων ανταγωνισμού, όπου planners με παρόμοιες δυνατότητες θα μπορούσαν να αξιολογηθούν σε προβλήματα αντίστοιχης φύσης. Η εξέλιξη αυτή δεν είχε ως στόχο απλώς την αύξηση της δυσκολίας, αλλά την προώθηση της έρευνας σε νέες, πιο εκφραστικές και πρακτικά χρήσιμες μορφές σχεδιασμού, αντικατοπτρίζοντας τις πραγματικές ανάγκες εφαρμογών σε τομείς όπως η ρομποτική και τα logistics.

3.2 Ο Διαγωνισμός IPC 2023: Μια Σύγχρονη Επισκόπηση

Έχοντας σκιαγραφήσει την ιστορική εξέλιξη του διαγωνισμού, η ανάλυση εστιάζει πλέον στην πιο πρόσφατη διοργάνωσή του, τον **IPC 2023**, που πραγματοποιήθηκε στο πλαίσιο του συνεδρίου ICAPS στην Πράγα. Η επισκόπηση αυτή είναι κρίσιμη, καθώς αποτυπώνει το τρέχον τοπίο της έρευνας και την κατάσταση της τέχνης (state-of-the-art) στα συστήματα σχεδιασμού, το οποίο αποτελεί και το πλαίσιο αναφοράς για την αξιολόγηση του planner που αναπτύχθηκε στην παρούσα εργασία. Στις υποενότητες που ακολουθούν, θα γίνει μια συνοπτική παρουσίαση των βασικών κατηγοριών (tracks) του διαγωνισμού, δίνοντας ιδιαίτερη έμφαση στο **Numeric Track**, το οποίο είναι άμεσα σχετικό με το πρόβλημα 'Rover' [22].

3.2.1 Τα Tracks του Διαγωνισμού

Ως αποτέλεσμα της εξέλιξης που περιγράφηκε στην προηγούμενη ενότητα, ο διαγωνισμός IPC 2023 διοργανώθηκε σε πολλαπλές, διακριτές κατηγορίες, καθεμία από τις οποίες εστιάζει σε μια διαφορετική επιστημονική πρόκληση. Οι κυριότερες κατηγορίες (tracks) που έλαβαν χώρα ήταν οι εξής:

- **Classical Track**: Ο ιστορικός πυρήνας του διαγωνισμού, εξετάζει προβλήματα με λογικές προϋποθέσεις και ντετερμινιστικά αποτελέσματα. Διακρίνεται σε υποκατηγορίες βέλτιστης (optimal) και ικανοποιητικής (satisficing) σχεδίασης [23].

- **Numeric Track**: Βρίσκεται στο επίκεντρο της παρούσας εργασίας και εξετάζει προβλήματα όπου οι δράσεις εξαρτώνται από αριθμητικές μεταβλητές (numeric fluents), όπως η διαχείριση πόρων[24].

- **HTN Track**: Εδώ, οι planners πρέπει να αποσυνθέσουν αφηρημένες, ιεραρχικά δομημένες εργασίες σε εκτελέσιμες δράσεις, αντί να βρίσκουν απλώς μια γραμμική ακολουθία [25].

- **Probabilistic Track**: Αντιμετωπίζει την αβεβαιότητα στα αποτελέσματα των δράσεων. Οι planners σε αυτό το track πρέπει να δημιουργήσουν πολιτικές (policies) που μεγιστοποιούν την πιθανότητα επιτυχίας ή την αναμενόμενη ανταμοιβή [26].

- **Learning Track**: Εισάγει την πρόκληση της ατελούς γνώσης του μοντέλου, όπου οι planners καλούνται να μάθουν τους κανόνες του προβλήματος μέσα από την αλληλεπίδραση με ένα προσομοιωμένο περιβάλλον [27].

Αυτή η εξειδίκευση επιτρέπει τη δίκαιη σύγκριση συστημάτων με παρόμοιες δυνατότητες και την ανάδειξη της προόδου σε κάθε επιμέρους υποπεδίο της σχεδίασης. Στον παρακάτω πίνακα μπορείτε να δείτε βασικές διαφορές των διάφορων tracks.

Πίνακας 3.1: Σύνοψη των κυριότερων tracks του IPC 2023.

Track	**Είδος Προβλήματος**	**Γλώσσα Μοντελοποίησης**	**Κριτήριο Αξιολόγησης**
Classical	Λογικό, Ντετερμινιστικό	PDDL	Ποιότητα πλάνου / Ταχύτητα
Numeric	Αριθμητικό, Ντετερμινιστικό	PDDL με numeric fluents	Ποιότητα πλάνου / Ταχύτητα
HTN	Ιεραρχική Αποσύνθεση	HDDL	Αριθμός επιλυμένων προβλημάτων
Probabilistic	Πιθανοτικό, Στοχαστικό	PPDDL / RDDL	Αναμενόμενη ανταμοιβή / Πιθανότητα επιτυχίας
Learning	Άγνωστο Μοντέλο	PDDL (για το τελικό πρόβλημα)	Ακρίβεια μοντέλου / Ποιότητα πλάνου

3.2.2 Το Numeric Track του IPC 2023

Η κατηγορία που σχετίζεται άμεσα με την παρούσα πτυχιακή εργασία είναι το **Numeric Track**. Στόχος του είναι η αξιολόγηση planners που μπορούν να διαχειριστούν προβλήματα με αριθμητικές μεταβλητές και περιορισμούς, τα οποία είναι ιδιαίτερα διαδεδομένα σε ρεαλιστικές εφαρμογές όπως η ρομποτική και η διαχείριση πόρων.

Για να αξιολογηθούν διαφορετικές πτυχές της απόδοσης, το Numeric Track του 2023 χωρίστηκε σε τρεις υποκατηγορίες:

- **Agile**:Σε αυτή την κατηγορία δίνεται ένας πολύ μικρός χρόνος (π.χ., 30 δευτερόλεπτα) για την εύρεση μιας λύσης, δίνοντας έμφαση στην ταχύτητα και την ικανότητα γρήγορης εύρεσης ενός αρχικού, έγκυρου πλάνου.
- **Satisficing**: Εδώ, ο στόχος είναι η εύρεση ενός πλάνου όσο το δυνατόν καλύτερης ποιότητας (χαμηλότερου κόστους) μέσα σε ένα τυπικό χρονικό όριο (π.χ., 30 λεπτά).
- **Optimal**: Αυτή είναι η πιο απαιτητική κατηγορία, όπου οι planners πρέπει να βρουν και να αποδείξουν ότι έχουν βρει ένα πλάνο με το ελάχιστο δυνατό

κόστος. Η παρούσα εργασία ευθυγραμμίζεται με τους στόχους αυτής της κατηγορίας, καθώς εστιάζει στην εύρεση βέλτιστων λύσεων.

Στην κατηγορία Optimal, όπου η παραδεκτότητα της ευρετικής είναι κρίσιμη, συμμετείχαν συστήματα που βασίζονται σε προηγμένες τεχνικές. Ένα παράδειγμα είναι ο planner **NLM-CutPlan**, ο οποίος αξιοποιεί μια ευρετική βασισμένη σε "τομές" (cuts) από γραμμικό προγραμματισμό για να εκτιμήσει το κόστος προς τον στόχο [28]. **Ένα από τα πιο εμβληματικά και διαχρονικά προβλήματα που χρησιμοποιήθηκαν και σε αυτόν τον διαγωνισμό για την αξιολόγηση των numeric planners είναι το πεδίο 'Rover', το οποίο θα αναλυθεί λεπτομερώς στην επόμενη ενότητα.**

3.3 Το Πεδίο Προβλημάτων "Rover" (The Rover Domain)

Αφού παρουσιάστηκε το γενικό πλαίσιο αξιολόγησης των planners (IPC) και η κατηγορία που μας ενδιαφέρει (Numeric Track), η ανάλυση πλέον εστιάζει στο συγκεκριμένο πεδίο προβλημάτων που αποτελεί τον πυρήνα της παρούσας πτυχιακής: το πεδίο "Rover". Το "Rover" δεν επιλέχθηκε τυχαία. Αποτελεί ένα από τα πιο κλασικά και αναγνωρίσιμα benchmarks στο numeric planning, το οποίο, όπως θα δούμε, συνδυάζει ενδιαφέρουσες λογικές και αριθμητικές προκλήσεις. Στις υποενότητες που ακολουθούν, θα εξετάσουμε την ιστορία και το πλαίσιο του domain, θα αναλύσουμε λεπτομερώς τη δομή του όπως αυτή ορίζεται στη γλώσσα PDDL, και θα συνοψίσουμε τις βασικές προκλήσεις που το καθιστούν ένα ιδανικό πεδίο για τη συγκριτική μελέτη εξειδικευμένων και γενικών planners.

3.3.1 Ιστορικό και Πλαίσιο

Το πεδίο προβλημάτων 'Rover' αποτελεί ένα από τα πιο κλασικά και διαχρονικά παραδείγματα στον χώρο του αυτοματοποιημένου σχεδιασμού. Η δημιουργία του εμπνεύστηκε άμεσα από τις πραγματικές προκλήσεις που αντιμετώπιζαν οι αποστολές εξερεύνησης πλανητών της NASA, όπως οι αποστολές Mars Exploration Rover (MER) του 2003 και η αποστολή Mars Science Laboratory (MSL) του 2009. Εμφανίστηκε για πρώτη φορά στον 3ο Διεθνή Διαγωνισμό Σχεδιασμού (IPC-3) το 2002, όπου και καθιερώθηκε ως ένα απαιτητικό benchmark για planners που χειρίζονται αριθμητικούς πόρους [3].

Παρά την ηλικία του, το 'Rover' παραμένει ένα εξαιρετικά επίκαιρο και χρήσιμο πεδίο για την έρευνα για διάφορους λόγους. Πρώτον, επιτυγχάνει μια εξαιρετική ισορροπία μεταξύ **λογικής πολυπλοκότητας** (π.χ., η αλληλουχία δράσεων για τη λήψη μιας φωτογραφίας) και **αριθμητικών περιορισμών** (η διαχείριση της ενέργειας). Δεύτερον, το πρόβλημα είναι διαισθητικά κατανοητό, καθώς οι στόχοι και οι περιορισμοί του αντιστοιχούν σε πραγματικές, απτές έννοιες, κάνοντάς το ιδανικό για την ανάλυση και την οπτικοποίηση των παραγόμενων

πλάνων. Τέλος, το πρόβλημα είναι εύκολα κλιμακούμενο (scalable): αυξάνοντας τον αριθμό των οχημάτων, των σημείων ενδιαφέροντος και των στόχων, η δυσκολία του προβλήματος αυξάνεται εκθετικά, επιτρέποντας τη δοκιμή των ορίων ακόμη και των πιο σύγχρονων planners. Για αυτούς τους λόγους, το 'Rover' συνεχίζει να χρησιμοποιείται ως ένα "legacy" benchmark, προσφέροντας ένα σταθερό σημείο αναφοράς για την πρόοδο του πεδίου.

3.3.2 Ανάλυση του Domain σε PDDL

Για την πλήρη κατανόηση της δομής και της πολυπλοκότητας του προβλήματος, είναι απαραίτητη η ανάλυση της επίσημης περιγραφής του πεδίου, όπως αυτή ορίζεται στο αρχείο $domain.pddl$. Η ανάλυση που ακολουθεί βασίζεται στην έκδοση του domain που χρησιμοποιήθηκε στον IPC 2023 [29].

1. **Τύποι Αντικειμένων και Κατηγορήματα**

   Το πεδίο ορίζει ένα σύνολο από τύπους αντικειμένων που αναπαριστούν τις βασικές οντότητες του προβλήματος: $rover$, $lander$, $objective$, $camera$, $mode$, $waypoint$, και $store$. Οι σχέσεις μεταξύ αυτών των αντικειμένων και οι ιδιότητές τους περιγράφονται από μια σειρά κατηγορημάτων (predicates), τα σημαντικότερα των οποίων είναι:

   - **Κατηγορήματα Τοπολογίας και Πλοήγησης**: Ορίζουν τον χάρτη του περιβάλλοντος και τις δυνατότητες κίνησης , π.χ.
     - $(in ?r - rover ?w - waypoint)$: Δηλώνει ότι το όχημα $?r$ βρίσκεται στην τοποθεσία $?w$.
     - $(can\_traverse ?r ?x ?y)$: Δηλώνει ότι το όχημα $?r$ μπορεί να διασχίσει τη διαδρομή μεταξύ των τοποθεσιών $?x$ και $?y$.
     - $(visible ?w1 ?w2)$: Δηλώνει ότι υπάρχει οπτική επαφή μεταξύ των τοποθεσιών $?w1$ και $?w2$, μια προϋπόθεση για την πλοήγηση και την επικοινωνία.
   - **Κατηγορήματα Εξοπλισμού και Δυνατοτήτων**: Καθορίζουν τις δυνατότητες του κάθε rover, π.χ.
     - $(equipped\_for\_soil\_analysis ?r)$: Δηλώνει ότι το rover είναι εξοπλισμένο για ανάλυση εδάφους.
     - $(on\_board ?c ?r)$: Δηλώνει ότι η κάμερα $?c$ βρίσκεται πάνω στο rover $?r$.
     - $(supports ?c ?m)$: Δηλώνει ότι η κάμερα $?c$ υποστηρίζει τη λειτουργία (mode) $?m$.
   - **Κατηγορήματα Κατάστασης Αποστολής**: Παρακολουθούν την πρόοδο προς την επίτευξη των στόχων π.χ.

- $(at\_soil\_sample ?w)$: Δηλώνει ότι στην τοποθεσία $?w$ υπάρχει διαθέσιμο δείγμα εδάφους.
- $(have\_rock\_analysis ?r ?w)$: Δηλώνει ότι το rover $?r$ έχει ήδη μια ανάλυση πετρώματος από την τοποθεσία $?w$.
- $(communicated\_image\_data ?o ?m)$: Δηλώνει ότι τα δεδομένα μιας εικόνας του στόχου $?o$ στη λειτουργία $?m$ έχουν ήδη μεταδοθεί.

2. **Αριθμητικά Χαρακτηριστικά και Συνάρτηση Κόστους**

   Τα αριθμητικά στοιχεία του 'Rover' είναι καθοριστικά για τη φύση του. Το πεδίο ορίζει δύο βασικές αριθμητικές συναρτήσεις:

   - $(energy ?r - rover)$: Αυτή η μεταβλητή λειτουργεί ως **αναλώσιμος πόρος** . Παρακολουθεί το επίπεδο της μπαταρίας κάθε οχήματος. Σχεδόν κάθε δράση στο πεδίο έχει ενεργειακό κόστος, και η τιμή της ενέργειας λειτουργεί ως προϋπόθεση για την εκτέλεση των περισσότερων δράσεων.
   - $(recharges)$: Πέρα από την ενέργεια, το πεδίο ορίζει και μια δεύτερη, καθολική αριθμητική συνάρτηση. Αυτή η συνάρτηση δεν λειτουργεί ως πόρος, αλλά ως **μετρητής κόστους**. Η τιμή της αυξάνεται κατά μία μονάδα $((increase (recharges) 1))$ κάθε φορά που εκτελείται η δράση recharge. Σε πολλά προβλήματα του IPC, ο στόχος είναι η **ελαχιστοποίηση (minimize)** της τελικής τιμής αυτής της συνάρτησης. Αυτό εισάγει έναν επιπλέον, πολύ ενδιαφέροντα συμβιβασμό: ο planner πρέπει να βρει μια λύση που όχι μόνο είναι εφικτή ενεργειακά, αλλά το κάνει και με τις **λιγότερες δυνατές επαναφορτίσεις**.

3. **Βασικές Δράσεις (Actions)**

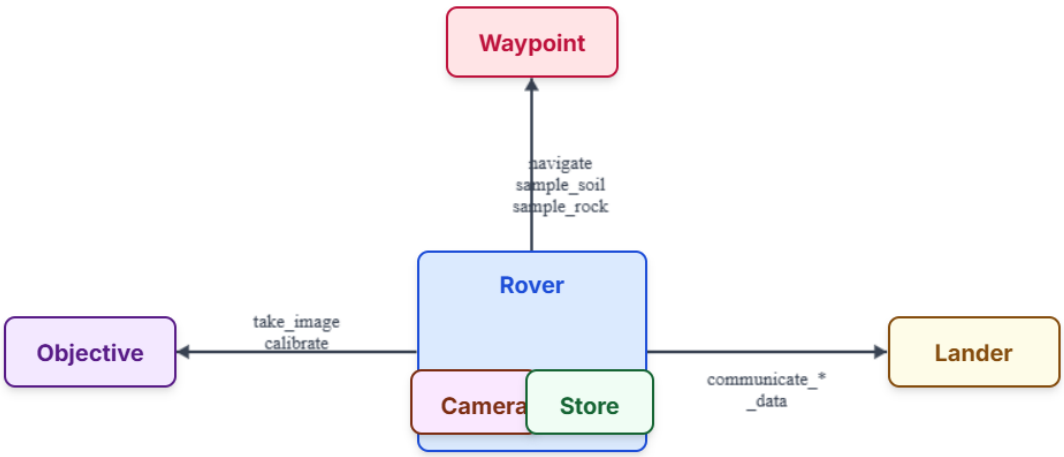
   Η δυναμική του προβλήματος ορίζεται από τις δράσεις. Οι πιο σημαντικές είναι:

   - $Navigate$:
     - **Σκοπός**: Μετακίνηση ενός rover μεταξύ δύο ορατών waypoints.
     - **Βασικές Προϋποθέσεις**: Το όχημα μπορεί να διασχίσει τη διαδρομή, υπάρχει οπτική επαφή και, κυρίως, επαρκής ενέργεια ($>=$ $(energy ...) ...$).
     - **Κύρια Αποτελέσματα**: Αλλαγή της θέσης του rover $(at ...)$ και μείωση της ενέργειας $(decrease (energy ...))$.
   - $sample\_soil$ / $sample\_rock$:
     - **Σκοπός**: Λήψη δειγμάτων εδάφους ή πετρωμάτων.

- **Βασικές Προϋποθέσεις**: Το rover πρέπει να βρίσκεται στη σωστή τοποθεσία, να έχει τον κατάλληλο εξοπλισμό, ο αποθηκευτικός του χώρος (*store*) να είναι άδειος (*empty*) και να έχει επαρκή ενέργεια.
- **Κύρια Αποτελέσματα**: Καταναλώνουν ενέργεια, γεμίζουν τον αποθηκευτικό χώρο (*full*), αποκτούν την ανάλυση (*have_..._analysis*) και αφαιρούν το δείγμα από τον χάρτη (*not*(*at_..._sample...*)).

- *recharge*:
  - **Σκοπός**: Επαναφόρτιση της μπαταρίας του οχήματος.
  - **Βασικές Προϋποθέσεις**: Το rover πρέπει να βρίσκεται σε μια τοποθεσία που βρίσκεται στον ήλιο (*in_sun*).
  - **Κύρια Αποτελέσματα**: Είναι η μοναδική δράση που **αυξάνει** (*increase*) την ενέργεια, ενώ ταυτόχρονα **αυξάνει** και τον μετρητή κόστους (*recharges*).

- *calibrate*/*take*$_i$*mage*/*communicate_*$*$:
  - **Σκοπός**: Ένα σύνολο δράσεων για τη φωτογράφιση στόχων και την επικοινωνία των δεδομένων.
  - **Βασικές Προϋποθέσεις**: Απαιτούν σύνθετες λογικές αλληλουχίες (π.χ., η *take_image* απαιτεί *calibrated* κάμερα) και έχουν σημαντικό ενεργειακό κόστος.
  - **Κύρια Αποτελέσματα**: Καταναλώνουν ενέργεια και αλλάζουν την κατάσταση της αποστολής (π.χ., *have_image*, *communicated_...*).



Σχήμα 3.1: Εννοιολογικό Διάγραμμα του Domain 'Rover'.

Πίνακας 3.2: Σύνοψη Βασικών Δράσεων και του Ενεργειακού τους Αντίκτυπου στο Πεδίο 'Rover'.

Δράση	**Σύντομη Περιγραφή**	**Ενεργειακό Effect**
`navigate`	Μετακινεί το όχημα σε μια γειτονική τοποθεσία.	-8
`recharge`	Επαναφορτίζει την μπαταρία του οχήματος.	+20
`sample_soil`	Λαμβάνει δείγμα εδάφους από την τρέχουσα τοποθεσία.	-3
`sample_rock`	Λαμβάνει δείγμα πετρώματος από την τρέχουσα τοποθεσία.	-5
`calibrate`	Βαθμονομεί την κάμερα για έναν συγκεκριμένο στόχο.	-2
`take_image`	Τραβάει μια φωτογραφία ενός στόχου από μια ορατή τοποθεσία.	-1
`communicate_*`	Μεταδίδει δεδομένα (εδάφους, πετρώματος, εικόνας) στον σταθμό βάσης.	-4 ή -6 (αν πρόκειται για image)

3.3.3 Οι Προκλήσεις του Rover Domain

Η ανάλυση του πεδίου 'Rover' αναδεικνύει ότι η δυσκολία του δεν πηγάζει από ένα μεμονωμένο χαρακτηριστικό, αλλά από τον συνδυασμό δύο διακριτών μορφών πολυπλοκότητας: της λογικής και της αριθμητικής.

Η **λογική πολυπλοκότητα** πηγάζει από τις αυστηρές εξαρτήσεις μεταξύ των δράσεων. Για παράδειγμα, για να μεταδοθεί μια εικόνα ($communicate\_image\_data$), πρέπει πρώτα να έχει ληφθεί ($take\_image$), κάτι που με τη σειρά του προϋποθέτει ότι η κάμερα έχει βαθμονομηθεί ($calibrate$), το όχημα βρίσκεται σε κατάλληλη ορατή τοποθεσία ($visible\_from$), και διαθέτει τον απαραίτητο εξοπλισμό ($equipped\_for\_imaging$). Η εύρεση της σωστής αλληλουχίας αυτών των δράσεων για την επίτευξη πολλαπλών και συχνά αλληλοεξαρτώμενων στόχων αποτελεί ένα κλασικό πρόβλημα σχεδιασμού από μόνο του [1].

Η **αριθμητική πολυπλοκότητα**, ωστόσο, είναι αυτή που καθιστά το πρόβλημα ιδιαίτερα απαιτητικό για βέλτιστους planners. Όπως αναλύθηκε, σχεδόν κάθε χρήσιμη δράση καταναλώνει ενέργεια. Αυτό δημιουργεί έναν κρίσιμο και συνεχή **συμβιβασμό (trade-off)**: το όχημα πρέπει να αποφασίσει αν θα συνεχίσει την εξερεύνηση και την εκτέλεση επιστημονικών στόχων, με τον κίνδυνο να εξαντληθεί η ενέργειά του, ή αν θα διακόψει την πορεία του για να μεταβεί σε

ένα σημείο επαναφόρτισης. Αυτή η απόφαση γίνεται ακόμα πιο σύνθετη από τη συνάρτηση κόστους $(recharges)$, η οποία τιμωρεί κάθε πράξη επαναφόρτισης. Επομένως, ένα βέλτιστο πλάνο δεν είναι απλώς αυτό που ολοκληρώνει τους στόχους, αλλά αυτό που το κάνει ελαχιστοποιώντας τις επαναφορτίσεις, βρίσκοντας την ιδανική ισορροπία μεταξύ της κατανάλωσης και της παραγωγής ενέργειας.

Συμπερασματικά, οι προκλήσεις του 'Rover' θέτουν τις βάσεις για τη μεθοδολογία που θα ακολουθηθεί. Η **λογική πολυπλοκότητα** και το μεγάλο μέγεθος του χώρου καταστάσεων καθιστούν αναγκαία τη χρήση ενός ισχυρού αλγορίθμου ευρετικής αναζήτησης, όπως ο **A***, ο οποίος μπορεί να εξερευνήσει αποδοτικά τον χώρο προς τη βέλτιστη λύση. Από την άλλη πλευρά, η **αριθμητική πολυπλοκότητα** –και ειδικότερα ο κρίσιμος συμβιβασμός της διαχείρισης ενέργειας– δεν μπορεί να αντιμετωπιστεί αποτελεσματικά από μια γενική ευρετική. Απαιτείται ο σχεδιασμός **εξειδικευμένων, domain-dependent ευρετικών συναρτήσεων** που θα ενσωματώνουν τη γνώση του προβλήματος, εκτιμώντας με ακρίβεια το μελλοντικό κόστος των αποφάσεων (π.χ., το κόστος μιας επερχόμενης επαναφόρτισης). Ο συνδυασμός του A* με αυτές τις custom ευρετικές αποτελεί τον πυρήνα της προσέγγισης που αναλύεται στο επόμενο κεφάλαιο.

Κεφάλαιο 4

Μεθοδολογία και Αποτελέσματα

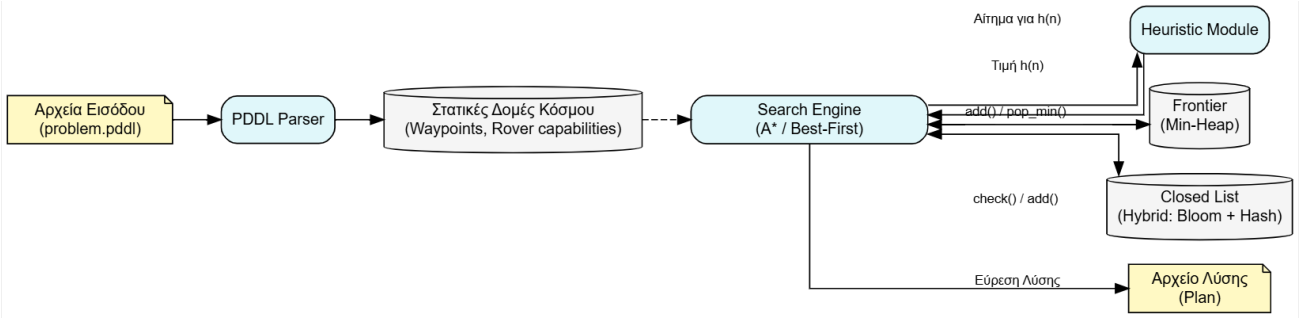
4.1 Επισκόπηση της Υλοποιούμενης Προσέγγισης

Όπως αναλύθηκε στο προηγούμενο κεφάλαιο, το πεδίο 'Rover' συνδυάζει λογική και αριθμητική πολυπλοκότητα, καθιστώντας την εύρεση βέλτιστων λύσεων μια σημαντική πρόκληση. Για την αντιμετώπιση αυτής της πρόκλησης, αναπτύχθηκε ένας εξειδικευμένος, domain-dependent planner. Αυτή η προσέγγιση επιτρέπει την ενσωμάτωση βαθιάς γνώσης για τις ιδιαιτερότητες του προβλήματος, με στόχο την καθοδήγηση της αναζήτησης πολύ πιο αποτελεσματικά από ό,τι θα μπορούσε ένας planner γενικής χρήσης.

Η αρχιτεκτονική του planner που υλοποιήθηκε είναι αρθρωτή (modular) και ακολουθεί τις κλασικές αρχές των συστημάτων που βασίζονται σε ευρετική αναζήτηση. Αποτελείται από διακριτά στοιχεία που συνεργάζονται για την εύρεση της λύσης, όπως απεικονίζεται στο Σχήμα 4.1. Τα βασικά **δομικά στοιχεία** είναι: ο **Parser**, υπεύθυνος για την ανάγνωση και μετατροπή των αρχείων εισόδου PDDL σε εσωτερικές δομές δεδομένων, ο κεντρικός **Search Engine**, που υλοποιεί τον αλγόριθμο A* (ή πρώτα στο καλύτερο), το **Heuristic Module**, το οποίο παρέχει την ευρετική εκτίμηση για κάθε κατάσταση, και οι δομές για την **Αναπαράσταση Καταστάσεων**, που αποθηκεύουν όλες τις απαραίτητες πληροφορίες. Στις ενότητες που ακολουθούν, κάθε ένα από αυτά τα δομικά στοιχεία θα αναλυθεί λεπτομερώς.

4.2 Υλοποίηση του Planner

Αφού παρουσιάστηκε η γενική αρχιτεκτονική του συστήματος, η παρούσα ενότητα εμβαθύνει στις τεχνικές λεπτομέρειες της υλοποίησης κάθε δομικού στοιχείου. Θα εξετάσουμε πώς οι αφηρημένες έννοιες της PDDL μεταφράζονται σε συγκεκριμένες δομές δεδομένων στη γλώσσα C, πώς υλοποιήθηκε ο κεντρικός αλγόριθμος αναζήτησης A*, και ποιοι μηχανισμοί χρησιμοποιήθηκαν για την αποδοτική διαχείριση του χώρου αναζήτησης. Η ανάλυση βασίζεται στον πη-



Σχήμα 4.1: Αρχιτεκτονική του Planner.

γαίο κώδικα που παρασχέθηκε, ο οποίος είναι πλήρως διαθέσιμος σε δημόσιο repository στο GitHub [1].

4.2.1 PDDL Parser

Σε αντίθεση με τους domain-independent planners που πρέπει να αναλύουν τόσο το αρχείο του πεδίου όσο και του προβλήματος, η εξειδικευμένη φύση του planner μας απλοποιεί σημαντικά αυτή τη διαδικασία. Η δομή του πεδίου 'Rover'—δηλαδή οι τύποι, τα κατηγορήματα και οι κανόνες των δράσεων—είναι ήδη ενσωματωμένη και "σκληρά κωδικοποιημένη" (hard-coded) στη λογική του planner.

Συνεπώς, ο ρόλος του **Parser**, του οποίου η διεπαφή ορίζεται στο αρχείο $parser.h$ του repository, είναι στοχευμένος: αναλαμβάνει αποκλειστικά την ανάγνωση του αρχείου του προβλήματος ($problem.pddl$). Κατά την εκτέλεση, ο parser αναλύει το αρχείο αυτό για να εξάγει τις πληροφορίες που ορίζουν τη συγκεκριμένη περίπτωση προς επίλυση. Αυτές περιλαμβάνουν:

- **Τα αντικείμενα** (: $objects$): Τη λίστα με τα συγκεκριμένα waypoints, rovers, κάμερες, κ.λπ. που συμμετέχουν στη συγκεκριμένη αποστολή.
- **Την αρχική κατάσταση** (: $init$): Τα κατηγορήματα που είναι αρχικά αληθή (π.χ., οι αρχικές θέσεις των rovers) και τις αρχικές τιμές των αριθμητικών χαρακτηριστικών (π.χ., η αρχική ενέργεια).
- **Τον στόχο** (: $goal$): Τις συνθήκες που πρέπει να ικανοποιηθούν για την επιτυχή ολοκλήρωση της αποστολής.

Η υλοποίηση του parser βασίζεται στην ακολουθιακή (sequential) ανάγνωση του αρχείου προβλήματος. Ο κώδικας διατρέχει το αρχείο, αναζητώντας τις

[1] Το GitHub repository μπορείτε να το βρείτε στον ακόλουθο σύνδεσμο: https://github.com/nikts27/thesis-rover-planner.git

λέξεις-κλειδιά : *objects*, : *init*, και : *goal* για να αναγνωρίσει τις αντίστοιχες ενότητες. Για την εξαγωγή των δεδομένων, χρησιμοποιούνται συναρτήσεις χειρισμού συμβολοσειρών της C (όπως *strtok*, *strstr*) για τον εντοπισμό και την απομόνωση των ονομάτων των αντικειμένων και των κατηγορημάτων, αγνοώντας τους περιττούς χαρακτήρες όπως παρενθέσεις και κενά. Κάθε αναγνωρισμένη οντότητα (π.χ., ένα waypoint) ή μια αρχική συνθήκη (π.χ., η αρχική θέση ενός rover) χρησιμοποιείται για να αρχικοποιήσει τις αντίστοιχες δομές δεδομένων (*structs*) που ορίζονται στο *auxiliary.h*.

Το τελικό αποτέλεσμα είναι η διαμόρφωση της αρχικής κατάστασης και του στόχου στη μνήμη του προγράμματος. Ουσιαστικά, ο parser "ρυθμίζει" το ήδη υπάρχον μοντέλο του κόσμου με τις παραμέτρους της συγκεκριμένης αποστολής, καθιστώντας το έτοιμο για να το παραλάβει ο αλγόριθμος αναζήτησης που χρησιμοποιούμε κάθε φορά.

4.2.2 Αναπαράσταση του Κόσμου και των Καταστάσεων

Η αποδοτική αναπαράσταση του προβλήματος στη μνήμη είναι κρίσιμη για την απόδοση του planner. Στην υλοποίησή μας, η καρδιά της αναζήτησης είναι η κεντρική δομή *struct State*, η οποία περιέχει ένα πλήρες "στιγμιότυπο" ολόκληρου του κόσμου του προβλήματος σε μια δεδομένη χρονική στιγμή.

Σε αντίθεση με μια προσέγγιση όπου η κατάσταση θα περιείχε μόνο τις μεταβλητές, εδώ η *struct State* περιέχει **πλήρεις πίνακες από τις δομές** (*structs*) **που αναπαριστούν κάθε αντικείμενο** του προβλήματος, όπως ορίζονται στο *auxiliary.h*:

- $Rover\ rovers[MAX\_ROVERS]$
- $Waypoint\ waypoints[MAX\_WAYPOINTS]$
- $Camera\ cameras[MAX\_CAMERAS]$
- κ.ο.κ. για όλα τα αντικείμενα.

Όλες οι δυναμικές πληροφορίες—αυτές δηλαδή που αλλάζουν από δράση σε δράση—αποθηκεύονται ως μέλη (members) των αντίστοιχων δομών των αντικειμένων. Για παράδειγμα:

- Η τρέχουσα ενέργεια ενός οχήματος δεν είναι ένας ξεχωριστός πίνακας, αλλά το πεδίο *energy* μέσα στη δομή *struct Rover* του συγκεκριμένου οχήματος.
- Ομοίως, το αν υπάρχει διαθέσιμο δείγμα εδάφους σε ένα waypoint δεν είναι μια καθολική μεταβλητή, αλλά το πεδίο *has_soil_sample* μέσα στη δομή struct Waypoint του συγκεκριμένου waypoint.

Η υλοποίηση χρησιμοποιεί έξυπνα bitmasks (π.χ., *has_soil_analysis* στο *struct Rover* ή *visible_waypoints* στο *struct Waypoint*) για τη συμπιεσμένη αποθήκευση πολλαπλών λογικών συνθηκών εντός της δομής κάθε αντικειμένου. Με αυτόν τον τρόπο, κάθε αντίγραφο της *struct State* είναι αυτόνομο και περιέχει όλες τις πληροφορίες που απαιτούνται για την αξιολόγηση της συγκεκριμένης κατάστασης.

Είναι σημαντικό να γίνει η διάκριση μεταξύ της "κατάστασης" (*struct State*) και του "κόμβου αναζήτησης" (*struct tree_node*). Ο κόμβος αναζήτησης είναι η δομή που χρησιμοποιεί ο αλγόριθμος αναζήτησης και περιέχει, εκτός από την ίδια την κατάσταση (*currState*), επιπλέον πληροφορίες απαραίτητες για την αναζήτηση: έναν δείκτη προς τον γονικό κόμβο (*parent*), το βάθος του κόμβου (*depth*) το κόστος για την επίτευξη του κόμβου (g), την ευρετική τιμή (h) και την συνολική τιμή f.

4.2.3 Η Υλοποίηση του Αλγορίθμου Αναζήτησης

Ο πυρήνας του planner είναι μια υλοποίηση του γενικού αλγορίθμου Best-First Search, όπως αυτός περιγράφηκε στο Κεφάλαιο 2. Αυτή η προσέγγιση επιλέχθηκε για την ευελιξία της, καθώς επιτρέπει την εύκολη εναλλαγή της στρατηγικής αναζήτησης, αλλάζοντας απλώς τη συνάρτηση αξιολόγησης που καθοδηγεί τον αλγόριθμο. Η παρούσα ενότητα αναλύει αυτή την υλοποίηση. Αρχικά, θα περιγραφεί η **κεντρική λογική** της μηχανής αναζήτησης—η διαχείριση δηλαδή των λιστών OPEN και CLOSED και η διαδικασία επέκτασης των κόμβων—η οποία είναι κοινή και για τις δύο μεθόδους. Στη συνέχεια, θα εξεταστούν οι **δύο συγκεκριμένες συναρτήσεις αξιολόγησης** που υλοποιήθηκαν, οι οποίες μετατρέπουν τη γενική μηχανή αναζήτησης είτε σε έναν "άπληστο" (greedy) planner είτε στον βέλτιστο αλγόριθμο A*.

Η Κεντρική Λογική Αναζήτησης

Η κεντρική λογική της μηχανής αναζήτησης, όπως υλοποιείται στο *planner.c*, ακολουθεί την κλασική αρχιτεκτονική ενός αλγορίθμου Best-First Search, ο οποίος περιστρέφεται γύρω από δύο βασικές δομές δεδομένων: τη λίστα ανοικτών και τη λίστα κλειστών κόμβων.

- **Η Λίστα Ανοικτών Κόμβων (Frontier / OPEN List)**: Πρόκειται για την καρδιά του αλγορίθμου. Στην υλοποίησή μας, είναι μια ελάχιστη σωρός (min-heap), όπως ορίζεται στο *minheap.h*, η οποία αποθηκεύει δείκτες σε κόμβους αναζήτησης (*struct tree_node*). Η δομή αυτή εγγυάται ότι ο κόμβος με τη μικρότερη τιμή f βρίσκεται πάντα στην κορυφή της σωρού, επιτρέποντας την εξαγωγή του σε χρόνο O(logn). Αυτό είναι κρίσιμο για την απόδοση του αλγορίθμου σε μεγάλους χώρους αναζήτησης.

- **Η Λίστα Κλειστών Κόμβων (CLOSED List)**: Αυτή η λίστα περιέχει όλες τις καταστάσεις που έχουν ήδη εξερευνηθεί. Υλοποιείται με τον υβριδικό μηχανισμό Bloom Filter και Hash Set, με σκοπό την ταχύτατη απόρριψη διπλότυπων καταστάσεων και την αποφυγή κύκλων στην αναζήτηση.

Η διαδικασία αναζήτησης, όπως φαίνεται και στον Αλγόριθμο 4.1, εκτυλίσσεται ως εξής:

1. Ο αλγόριθμος αρχικοποιείται τοποθετώντας τον κόμβο της αρχικής κατάστασης στο Frontier.

2. Εκτελείται ένας κεντρικός βρόχος επανάληψης **όσο** το Frontier δεν είναι άδειο.

3. Σε κάθε επανάληψη, εξάγεται ο κόμβος με τη μικρότερη τιμή f από το Frontier.

4. Γίνεται έλεγχος αν η κατάσταση του κόμβου ικανοποιεί τις συνθήκες-στόχο. Αν ναι, η αναζήτηση τερματίζει επιτυχώς και το μονοπάτι-λύση ανακατασκευάζεται χρησιμοποιώντας τους δείκτες προς τους γονικούς κόμβους, ώστε να γραφτεί σε αρχείο εξόδου.

5. Αν όχι, ο κόμβος προστίθεται στη λίστα κλειστών κόμβων και ακολουθεί η διαδικασία της **επέκτασης (node expansion)**.

Κατά την επέκταση, ο planner δημιουργεί όλους τους πιθανούς διαδόχους (successors) του τρέχοντος κόμβου. Αυτό γίνεται εφαρμόζοντας όλες τις έγκυρες δράσεις του πεδίου 'Rover' στην τρέχουσα κατάσταση. Για κάθε έγκυρη δράση (που ικανοποιεί δηλαδή όλα τα preconditions που απαιτεί), δημιουργείται μια νέα κατάσταση-παιδί ($State$) εφαρμόζοντας τα effect της, υπολογίζονται οι τιμές g και h για τον νέο κόμβο, και αυτός εισάγεται στο Frontier, εφόσον δεν έχει εξερευνηθεί ξανά.

Είναι σημαντικό να σημειωθεί ότι ο παρακάτω ψευδοκώδικας αποτελεί μια υψηλού επιπέδου αφαίρεση της συνολικής λογικής και όχι μια κατά γράμμα μεταφορά του κώδικα C. Στην πραγματική υλοποίηση, για λόγους αρθρωτότητας (modularity) και καθαρότητας του κώδικα, ορισμένες από τις παραπάνω λειτουργίες, όπως η δημιουργία των διαδόχων, είναι οργανωμένες σε ξεχωριστές βοηθητικές συναρτήσεις. Επίσης, ο ψευδοκώδικας αποκρύπτει σκόπιμα λεπτομέρειες χαμηλού επιπέδου που αφορούν τη διαχείριση της μνήμης και την απευθείας χρήση των βιβλιοθηκών, όπως την διαχείριση του Frontier. Σκοπός του είναι να παρουσιάσει με ακαδημαϊκή σαφήνεια την αλγοριθμική ροή, καθιστώντας την κατανοητή στον αναγνώστη.

Algorithm 3 Η Λογική Αναζήτησης του Planner

```
1: function SEARCH(initial_state, goal_description, method)
2:     Frontier ← new MinHeap()
3:     ClosedList ← new ClosedList()
4:     start_node ← create_node(initial_state, parent=NULL)
5:     start_node.g ← 0
6:     start_node.h ← calculate_heuristic(start_node.state)
7:     set_f_score(start_node, method) ▷ f = g+h για A*, f = h για Greedy best-first
8:     Frontier.add(start_node)
9:     while Frontier is not empty do
10:        current_node ← Frontier.pop_min()
11:        if ClosedList.contains(current_node.state) then
12:            continue              ▷ Αν ο κόμβος έχει ήδη εξερευνηθεί, αγνοείται
13:        end if
14:        if is_solution(current_node.state, goal_description) then
15:            return reconstruct_path(current_node)
16:        end if
17:        ClosedList.add(current_node.state)
18:        for each valid action from current_node.state do
19:            successor_state ← apply_action(current_node.state, action)
20:            if ClosedList.contains(successor_state) == FALSE then
21:                successor_node←create_node(successor_state,
    parent=current_node)
22:                successor_node.g ← current_node.g + cost(action)
23:                successor_node.h ← calculate_heuristic(successor_node.state)
24:                set_f_score(successor_node, method)
25:                Frontier.add(successor_node)
26:            end if
27:        end for
28:    end while
29:    return NULL
30: end function
```

Οι Συναρτήσεις Αξιολόγησης

Όπως περιγράφηκε στον ψευδοκώδικα, η συμπεριφορά της μηχανής αναζήτησης καθορίζεται από την παράμετρο $method$, η οποία επιλέγει τη συνάρτηση αξιολόγησης που θα χρησιμοποιηθεί για τον υπολογισμό του f_score κάθε κόμβου (ο χρήστης στην κλήση του προγράμματος επιλέγει τον αλγόριθμο που θέλει να εφαρμοστεί). Η υλοποίηση υποστηρίζει δύο διακριτές στρατηγικές:

- **Greedy Best-First Search**

  Όταν επιλέγεται αυτή η μέθοδος, ο planner αγνοεί το πραγματικό κόστος

$g(n)$ που έχει ήδη διανυθεί από την αρχική κατάσταση. Η συνάρτηση αξιολόγησης απλοποιείται σε:

$$f(n) = h(n)$$

Αυτή η προσέγγιση είναι "άπληστη" (greedy), καθώς κατευθύνει την αναζήτηση προς τον κόμβο που φαίνεται να είναι πιο κοντά στον στόχο, με βάση αποκλειστικά την ευρετική εκτίμηση. Οδηγεί συχνά σε λύσεις πολύ γρήγορα, αλλά δεν παρέχει καμία εγγύηση ότι η λύση που θα βρεθεί θα είναι η βέλτιστη (η φθηνότερη).

- **A* Search**

  Όταν επιλέγεται η μέθοδος A*, ο planner υιοθετεί την πλήρη συνάρτηση αξιολόγησης:

  $$f(n) = g(n) + h(n)$$

  Εδώ, ο αλγόριθμος εξισορροπεί το πραγματικό κόστος $g(n)$ για την επίτευξη ενός κόμβου με την εκτιμώμενη απόσταση $h(n)$ από αυτόν μέχρι τον στόχο. Όπως αναλύθηκε στο Κεφάλαιο 2, εφόσον η ευρετική συνάρτηση $h(n)$ είναι παραδεκτή (admissible), ο αλγόριθμος A* εγγυάται την εύρεση της βέλτιστης λύσης. Αυτή η στρατηγική είναι ο πυρήνας της αναζήτησης για βέλτιστα πλάνα στην παρούσα εργασία [9].

4.2.4 Διαχείριση Διπλότυπων Καταστάσεων

Ένα από τα μεγαλύτερα προβλήματα στην αναζήτηση σε γράφους είναι η πιθανότητα να επισκεφθεί ο αλγόριθμος την ίδια κατάσταση πολλαπλές φορές. Αυτό οδηγεί σε εκθετική αύξηση του χρόνου αναζήτησης και μπορεί να οδηγήσει σε ατέρμονους βρόχους. Για την αντιμετώπιση αυτού του φαινομένου, οι αλγόριθμοι αναζήτησης διατηρούν μια λίστα κλειστών κόμβων (**CLOSED list**), η οποία αποθηκεύει όλες τις καταστάσεις που έχουν ήδη εξερευνηθεί. Η αποδοτικότητα αυτού του μηχανισμού είναι κρίσιμη για την συνολική απόδοση του planner [1].

Στην παρούσα υλοποίηση, για τη διαχείριση του CLOSED list, αναπτύχθηκε ένας **υβριδικός μηχανισμός** που συνδυάζει δύο δομές δεδομένων με διαφορετικά χαρακτηριστικά, με στόχο τη βελτιστοποίηση της ταχύτητας και της χρήσης μνήμης.

1. **Bloom Filter (Πρώτο Επίπεδο Ελέγχου)**: Το πρώτο επίπεδο ελέγχου υλοποιείται με ένα **Bloom Filter**. Το Bloom filter είναι μια πιθανοτική δομή δεδομένων, εξαιρετικά αποδοτική σε χώρο, που μπορεί να απαντήσει

γρήγορα στην ερώτηση "έχει εμφανιστεί ξανά αυτό το στοιχείο;". Η απάντησή του μπορεί να είναι είτε "**σίγουρα όχι**" είτε "**ίσως ναι**", αλλά ποτέ "σίγουρα ναι". Αυτό σημαίνει ότι δεν έχει ψευδώς αρνητικά αποτελέσματα (false negatives), αλλά μπορεί να έχει ψευδώς θετικά (false positives)[2].

2. **Hash Set (Δεύτερο Επίπεδο Ελέγχου)**: Το δεύτερο επίπεδο είναι ένα απλό Hash Set. Αυτή είναι μια ντετερμινιστική δομή που εγγυάται 100 τοις εκατό ακρίβεια, αλλά με μεγαλύτερο κόστος σε μνήμη και χρόνο πρόσβασης σε σχέση με το Bloom filter[3].

Η συνδυαστική λειτουργία τους είναι η εξής: για κάθε νέα κατάσταση, ο planner ελέγχει πρώτα το Bloom filter. Αν η απάντηση είναι "σίγουρα όχι", η κατάσταση θεωρείται νέα και προστίθεται και στις δύο δομές. Αν η απάντηση είναι "ίσως ναι", μόνο τότε ο planner προχωρά στον πιο "ακριβό" έλεγχο στο Hash Set για την τελική επιβεβαίωση. Αυτή η στρατηγική μειώνει δραστικά τον αριθμό των προσπελάσεων στο Hash Set, βελτιώνοντας την ταχύτητα.

Πίνακας 4.1: Σύγκριση των Δομών του Υβριδικού Μηχανισμού Εντοπισμού Διπλότυπων.

Χαρακτηριστικό	**Hash Set**	**Bloom Filter**
Ακρίβεια	100% (Ντετερμινιστικό)	Πιθανοτικό
Χρήση Μνήμης	Υψηλή (αποθηκεύει ολόκληρα τα κλειδιά των καταστάσεων)	Πολύ Χαμηλή (αποθηκεύει μόνο έναν πίνακα από bits)
Ταχύτητα Ελέγχου	Γρήγορη (σταθερός χρόνος κατά μέσο όρο)	Εξαιρετικά Γρήγορη (μερικές απλές συναρτήσεις hash)
Πιθανότητα Σφάλματος	Καμία	Μικρή πιθανότητα για ψευδώς θετικά αποτελέσματα (false positives)

4.3 Ευρετικές Συναρτήσεις για το Rover Domain

Έχοντας περιγράψει την κεντρική μηχανή αναζήτησης του planner, η παρούσα ενότητα εστιάζει στο πιο κρίσιμο στοιχείο που καθοδηγεί την απόδοσή

[2]Ο κώδικας του bloom filter μπορεί να βρεθεί στον ακόλουθο σύνδεσμο: https://github.com/jvirkki/libbloom

[3]Ο κώδικας του hash set μπορεί να βρεθεί στον ακόλουθο σύνδεσμο: https://github.com/troydhanson/uthash

της: την ευρετική συνάρτηση. Σε αντίθεση με τους domain-independent planners, η εξειδικευμένη προσέγγισή μας επιτρέπει τον σχεδιασμό "στο χέρι" ευρετικών συναρτήσεων, ενσωματώνοντας βαθιά γνώση για τις ιδιαιτερότητες του πεδίου 'Rover'.

Στις υποενότητες που ακολουθούν, θα παρουσιαστεί μια σειρά από πέντε ευρετικές συναρτήσεις, η καθεμία από τις οποίες αποτελεί ένα βήμα στην εξελικτική διαδικασία σχεδιασμού. Ο παρακάτω πίνακας (Πίνακας 4.2) συνοψίζει τις ευρετικές που θα αναλυθούν, την κεντρική τους ιδέα και τη θεωρητική τους ιδιότητα όσον αφορά την παραδεκτότητα (admissibility).

Πίνακας 4.2: Σύνοψη των Ευρετικών Συναρτήσεων που Αναπτύχθηκαν.

Ευρετική	**Κεντρική Ιδέα**	**Ιδιότητα**
H0 (Zero)	Επιστρέφει πάντα 0.	**Admissible**
H1 (Sum)	Άθροισμα του κόστους όλων των ανεκπλήρωτων στόχων.	Non-admissible
H2 (Max)	Μέγιστο κόστος του πιο "ακριβού" μεμονωμένου στόχου.	**Admissible**
H3 (Sum-2)	Άθροισμα του κόστους των 2 πιο ακριβών στόχων από διαφορετικά rovers.	**Admissible**
H4 (Optimal Assign)	Άπληστη ανάθεση των πιο ακριβών στόχων σε όλα τα διαθέσιμα rovers.	**Admissible**

4.3.1 Ευρετική H0: Ευρετική Μηδέν (Admissible Baseline)

1. **Ορισμός και Σκεπτικό**

   Η πρώτη ευρετική που εξετάζεται είναι η απλούστερη δυνατή παραδεκτή ευρετική, η **Ευρετική Μηδέν**. Ο ορισμός της είναι εξαιρετικά απλός: για κάθε κατάσταση n, η ευρετική τιμή είναι πάντα μηδέν:

   $$h(n) = 0$$

   Ο σκοπός της δεν είναι να καθοδηγήσει την αναζήτηση, αλλά να δημιουργήσει μια **θεωρητική βάση αναφοράς (baseline)**. Όταν ο αλγόριθμος A* χρησιμοποιεί την $h(n) = 0$, η συνάρτηση αξιολόγησής του,$f(n) = g(n)+h(n)$, μετατρέπεται σε $f(n) = g(n)$. Αυτό καθιστά τον A* ισοδύναμο με τον αλγόριθμο **Αναζήτησης Ομοιόμορφου Κόστους (Uniform Cost Search)**, γνωστό και ως αλγόριθμος του Dijkstra. Τα αποτελέσματα από αυτή την

ευρετική θα μας δείξουν την απόδοση μιας "τυφλής", αλλά εγγυημένα βέλτιστης, αναζήτησης, έναντι της οποίας θα μετρηθούν όλες οι υπόλοιπες, πιο "έξυπνες" ευρετικές.

2. **Ανάλυση Παραδεκτότητας (Admissibility Analysis)**

   Παραδεκτότητα: Η ευρετική αυτή είναι **προφανώς παραδεκτή (trivially admissible)**. Η συνθήκη της παραδεκτότητας $(h(n) <= h'(n))$ ικανοποιείται πάντα, καθώς το πραγματικό κόστος $h'(n)$ για την επίτευξη του στόχου είναι πάντοτε μη-αρνητικό (αφού τα κόστη των δράσεων είναι θετικά). Συνεπώς, ισχύει πάντα ότι $0 <= h'(n)$. Αυτό εγγυάται ότι η αναζήτηση θα βρει τη βέλτιστη λύση.

3. **Υλοποίηση σε Ψευδοκώδικα**

   Η υλοποίηση της ευρετικής είναι η απλούστερη δυνατή:

Algorithm 4 Ευρετική Συνάρτηση H0: Ευρετική Μηδέν

```
1: function CALCULATE_HEURISTIC_H0(state)   ▷ state: Η τρέχουσα κατάσταση προς αξιολόγηση
2:     return 0                             ▷ Επιστρέφει πάντα 0, ανεξαρτήτως της κατάστασης
3: end function
```

4.3.2 Ευρετική H1: Άθροισμα Κόστους Στόχων (Sum - Non-admissible)

1. **Ορισμός και Σκεπτικό**

   Η πρώτη ουσιαστική ευρετική που αναπτύχθηκε, η $H1$ ή "**Sum Heuristic**", αποτελεί μια "άπληστη" (greedy) και διαισθητική προσέγγιση για την εκτίμηση του εναπομείναντος κόστους. Το κεντρικό σκεπτικό της είναι η αποσύνθεση του συνολικού προβλήματος στους επιμέρους ανεκπλήρωτους στόχους (unmet goals) και ο υπολογισμός του **αθροίσματος** των εκτιμώμενων "χαλαρών" κοστών για την επίτευξη του καθενός από αυτούς.

   Η βασική υπόθεση είναι ότι το συνολικό κόστος για την επίλυση του προβλήματος μπορεί να προσεγγιστεί αθροίζοντας το ελάχιστο κόστος που απαιτείται για κάθε μεμονωμένο στόχο, σαν να ήταν ανεξάρτητα υποπροβλήματα. Για κάθε ανεκπλήρωτο στόχο (π.χ., "communicate soil data from waypoint X"), η ευρετική υπολογίζει το φθηνότερο πλάνο για την επίτευξή του, λαμβάνοντας υπόψη το κόστος μετακίνησης, εκτέλεσης της δράσης και επικοινωνίας. Αυτή η προσέγγιση στοχεύει στο να δώσει μια πλούσια σε πληροφορία εκτίμηση, καθώς λαμβάνει υπόψη όλες τις εκκρεμείς εργασίες.

2. **Ανάλυση Παραδεκτότητας (Admissibility Analysis)**

   Μη-Παραδεκτότητα: Η ευρετική αυτή είναι **μη-παραδεκτή (non-admissible)**. Ο λόγος είναι ότι το απλό άθροισμα των "χαλαρών" κοστών για κάθε μεμονωμένο στόχο **αγνοεί πλήρως τις αλληλεπιδράσεις και τις συνέργειες** που μπορεί να υπάρχουν σε ένα πραγματικό πλάνο.

   Υπάρχουν δύο βασικά σενάρια όπου η ευρετική μπορεί να υπερεκτιμήσει το πραγματικό κόστος ($h(n) > h'(n)$):

   - **Κοινή Χρήση Πόρων**: Η ευρετική μπορεί να υπολογίσει ότι ο "Rover 1" είναι η φθηνότερη επιλογή για την επίτευξη του Στόχου Α και ταυτόχρονα η φθηνότερη επιλογή για την επίτευξη του Στόχου Β. Στη συνέχεια, αθροίζει αυτά τα δύο κόστη. Στην πραγματικότητα, όμως, ο "Rover 1" μπορεί να εκτελέσει και τους δύο στόχους σε ένα ενιαίο, βελτιστοποιημένο ταξίδι, με πολύ μικρότερο συνολικό κόστος από το άθροισμα των δύο μεμονωμένων.
   - **Κοινή Χρήση Υπο-πλάνων**: Ένα rover μπορεί να χρειαστεί να ταξιδέψει από το σημείο W1 στο W5 για να πετύχει τον Στόχο Α. Στη διαδρομή του, περνάει από το W3, όπου μπορεί να πετύχει και τον Στόχο Β με ελάχιστο επιπλέον κόστος. Η ευρετική sum θα αθροίσει το πλήρες κόστος για το ταξίδι προς τον Στόχο Α με το πλήρες κόστος για το ταξίδι προς τον Στόχο Β, υπερεκτιμώντας κατά πολύ το πραγματικό κόστος του βέλτιστου, συνδυαστικού πλάνου.

   Επειδή η ευρετική αυτή μπορεί να υπερεκτιμήσει το κόστος, ο A* που την χρησιμοποιεί δεν εγγυάται την εύρεση της βέλτιστης λύσης, αλλά στοχεύει στην ταχεία εύρεση μιας ικανοποιητικής λύσης.

3. **Υλοποίηση σε Ψευδοκώδικα**

   Η υλοποίηση της ευρετικής αποτελείται από τρία κύρια μέρη: τον υπολογισμό του κόστους για κάθε τύπο στόχου (soil, rock, image) και τον τελικό συνδυασμό τους. Ένα κρίσιμο προαπαιτούμενο είναι ο προ-υπολογισμός (precomputation) των συντομότερων διαδρομών μεταξύ όλων των waypoints για κάθε rover, ο οποίος γίνεται μία φορά στην αρχή με τον αλγόριθμο Floyd-Warshall.

Algorithm 5: Ευρετική Συνάρτηση H1: Sum Heuristic

```
1: function CALCULATE_HEURISTIC_H1(state)
2:     total_cost ← 0
           ▷ 1. Υπολογισμός κόστους για τους στόχους δειγμάτων εδάφους
3:     soil_cost ← Calculate_All_Soil_Goals_Cost(state)
4:     if soil_cost == ∞ then return ∞
5:     end if
```

```
    total_cost ← total_cost + soil_cost
                ▷ 2. Υπολογισμός κόστους για τους στόχους δειγμάτων πετρώματος
    rock_cost ← Calculate_All_Rock_Goals_Cost(state)
    if rock_cost == ∞ then return ∞
    end if
    total_cost ← total_cost + rock_cost
                ▷ 3. Υπολογισμός κόστους για τους στόχους εικόνων
    image_cost ← Calculate_All_Image_Goals_Cost(state)
    if image_cost == ∞ then return ∞
    end if
    total_cost ← total_cost + image_cost
            ▷ 4. Υπολογισμός του κόστους ενέργειας (επαναφορτίσεων)
    energy_cost ← Calculate_Energy_Deficit_Cost(state, total_cost)
    if energy_cost == ∞ then return ∞
    end if
    total_cost ← total_cost + energy_cost
    return total_cost
end function

function Calculate_Single_Goal_Cost(goal, state)
    min_cost_for_goal ← ∞
    for each rover in state do
        ▷ Υπολόγισε το "χαλαρό" κόστος για το 'goal' με αυτό το 'rover':
        cost ← cost_to_travel(rover, goal.location) + cost_to_execute(goal) + cost_to_travel_to_comm_point(rover, goal.location) + cost_to_communicate(goal)
        if cost < min_cost_for_goal then
            min_cost_for_goal ← cost
        end if
    end for
    return min_cost_for_goal
end function

function Calculate_Energy_Deficit_Cost(state, total_work_cost)
    total_recharge_cost ← 0
    work_per_rover ← ⌈ total_work_cost / num_rovers ⌉
    for each rover r in state do
        deficit ← work_per_rover - rover.energy
        if deficit > 0 then
            recharges_needed ← ⌈ deficit / 20 ⌉ ▷ 20 είναι η ενέργεια ανά επαναφόρτιση
            dist_to_sun ← find_min_dist_to_sunny_waypoint(rover.position)
```

```
            if dist_to_sun == ∞ then
                return ∞                                  ▷ Αδύνατη η επαναφόρτιση
            end if
            cost_for_rover ← dist_to_sun + recharges_needed
            total_recharge_cost ← total_recharge_cost + cost_for_rover
        end if
    end for
    return total_recharge_cost
end function
```

4.3.3 Ευρετική H2: Μέγιστο Κόστος Στόχου (Max - Admissible)

1. **Ορισμός και Σκεπτικό**

   Μετά τον εντοπισμό του θεωρητικού προβλήματος της Sum heuristic (ότι είναι μη-παραδεκτή), το επόμενο βήμα ήταν ο σχεδιασμός μιας ευρετικής συνάρτησης που θα εγγυάται τη βελτιστότητα. Η **H2**, ή "**Max Heuristic**" σχεδιάστηκε ακριβώς για αυτόν τον σκοπό.

   Το σκεπτικό της είναι απλό και βασίζεται στη θεμελιώδη ιδιότητα των παραδεκτών ευρετικών: πρέπει πάντα να υποεκτιμούν το πραγματικό κόστος. Αντί να αθροίζει τα κόστη όλων των ανεκπλήρωτων στόχων (κάτι που οδηγεί σε υπερεκτίμηση), η Max heuristic υπολογίζει το "χαλαρό" κόστος για κάθε μεμονωμένο ανεκπλήρωτο στόχο και επιστρέφει το **μέγιστο** από αυτά.

   Η λογική είναι η εξής: το πραγματικό κόστος για την επίλυση του συνολικού προβλήματος (h'(n)) είναι αναγκαστικά τουλάχιστον όσο το κόστος που απαιτείται για την επίλυση του πιο δύσκολου, μεμονωμένου του μέρους. Επομένως, το κόστος του πιο "ακριβού" ανεκπλήρωτου στόχου αποτελεί ένα ασφαλές κατώτατο όριο (a safe lower bound) για το κόστος ολόκληρης της λύσης.

2. **Ανάλυση Παραδεκτότητας (Admissibility Analysis)**

   Παραδεκτότητα: Η ευρετική αυτή είναι **εγγυημένα παραδεκτή (admissible)**.

   Η απόδειξη είναι ευθεία: Έστω $G_1, G_2, ..., Gk$ το σύνολο των ανεκπλήρωτων στόχων. Η ευρετική υπολογίζει το χαλαρό κόστος $c(Gi)$ για κάθε στόχο i και επιστρέφει $h(n) = max(c(G_1), c(G_2), ..., c(Gk))$. Έστω Gm ο στόχος με το μέγιστο κόστος, δηλαδή $h(n) = c(Gm)$. Το πραγματικό, βέλτιστο πλάνο για την επίλυση του συνολικού προβλήματος πρέπει *οπωσδήποτε* να πετύχει και τον στόχο Gm. Το πραγματικό κόστος για την επίτευξη του Gm είναι τουλάχιστον $c(Gm)$ (αφού το $c(Gm)$ είναι ήδη μια χαλαρή, αισιόδοξη εκτίμηση). Επομένως, το συνολικό πραγματικό κόστος $h'(n)$ για την επίτευξη όλων των στόχων θα είναι αναγκαστικά μεγαλύτερο ή ίσο από το κόστος για την επίτευξη του πιο δύσκολου μέρους του. Ισχύει δηλαδή:

$$h(n) = c(Gm) \leq h'(n)$$

Αυτό ικανοποιεί τη συνθήκη της παραδεκτότητας, εξασφαλίζοντας ότι ο A* θα βρει τη βέλτιστη λύση.

3. **Υλοποίηση σε Ψευδοκώδικα**

Η υλοποίηση ακολουθεί την ίδια λογική με την H1 για τον υπολογισμό του κόστους ενός μεμονωμένου στόχου, αλλά αντί να αθροίζει τα αποτελέσματα, κρατάει το μέγιστο. Ο υπολογισμός του energy deficit παραμένει ίδιος.

Algorithm 6 Ευρετική Συνάρτηση H2: Max Heuristic

```
1:  function CALCULATE_HEURISTIC_H2(state)
2:      max_work_cost ← 0
                ▷ 1. Εύρεση του max κόστους ανάμεσα στους στόχους εδάφους
3:      max_soil_cost ← Calculate_Max_Cost_For_Goal_Type(SOIL, state)
4:      max_work_cost ← max(max_work_cost, max_soil_cost)
              ▷ 2. Εύρεση του max κόστους ανάμεσα στους στόχους πετρώματος
5:      max_rock_cost ← Calculate_Max_Cost_For_Goal_Type(ROCK, state)
6:      max_work_cost ← max(max_work_cost, max_rock_cost)
                ▷ 3. Εύρεση του max κόστους ανάμεσα στους στόχους εικόνων
7:      max_image_cost ← Calculate_Max_Cost_For_Goal_Type(IMAGE, state)
8:      max_work_cost ← max(max_work_cost, max_image_cost)
9:      if max_work_cost == 0 then return 0
10:     end if                                  ▷ Όλοι οι στόχοι έχουν επιτευχθεί
          ▷ 4. Υπολογισμός του κόστους ενέργειας μόνο για τον πιο ακριβό στόχο
11:     energy_cost ← Calculate_Energy_Deficit_Cost(state, max_work_cost)
12:     if energy_cost == ∞ then return ∞
13:     end if
14:     return max_work_cost + energy_cost
15: end function

16: function CALCULATE_MAX_COST_FOR_GOAL_TYPE(type, state)
17:     max_cost ← 0
18:     for each unfulfilled_goal of the specified type do
19:         cost_for_this_goal ← Calculate_Single_Goal_Cost(unfulfilled_goal, state)
20:         max_cost ← max(max_cost, cost_for_this_goal)
21:     end for
22:     return max_cost
23: end function
```

4.3.4 Ευρετική H3: Άθροισμα Δύο Καλύτερων Στόχων (Sum-2-Goals - Admissible)

1. **Ορισμός και Σκεπτικό**

   Παρότι η Max heuristic (H2) έλυσε το πρόβλημα της παραδεκτότητας, η πειραματική ανάλυση έδειξε ότι είναι συχνά υπερβολικά "συντηρητική" ή "απαισιόδοξη". Η τιμή που επιστρέφει, αν και αποτελεί ένα ασφαλές κατώτατο όριο, είναι συχνά πολύ χαμηλή, παρέχοντας ανεπαρκή καθοδήγηση στον αλγόριθμο A*. Η βασική αιτία αυτής της αδυναμίας είναι ότι η Max **αγνοεί την εγγενή παραλληλία** που υπάρχει στο πεδίο 'Rover' όταν διατίθενται πολλαπλά οχήματα.

   Η **H3**, ή "**Sum-2-Goals Heuristic**", σχεδιάστηκε για να αντιμετωπίσει ακριβώς αυτή την αδυναμία. Το σκεπτικό της είναι να παρέχει μια πιο πληροφοριακή (και άρα υψηλότερη) ευρετική τιμή, εκμεταλλευόμενη την ιδέα ότι δύο διαφορετικά rovers μπορούν να εργάζονται ταυτόχρονα για την επίτευξη δύο διαφορετικών στόχων. Η ευρετική υπολογίζει το "χαλαρό" κόστος για όλους τους ανεκπλήρωτους στόχους και στη συνέχεια επιστρέφει το **άθροισμα του κόστους των δύο πιο "ακριβών" στόχων, υπό την κρίσιμη προϋπόθεση ότι αυτοί μπορούν να εκτελεστούν από δύο διαφορετικά rovers.**

2. **Ανάλυση Παραδεκτότητας (Admissibility Analysis)**

   Παραδεκτότητα: Η ευρετική αυτή είναι **παραδεκτή (admissible)**.

   Η λογική πίσω από την παραδεκτότητα έγκειται στην αρχή της **αποσύνθεσης του προβλήματος σε ανεξάρτητα υποπροβλήματα (disjoint subproblems)**. Έστω G_1 ο πιο ακριβός στόχος, που εκτελείται βέλτιστα από τον $Rover\ A$, και G_2 ο δεύτερος πιο ακριβός στόχος που εκτελείται βέλτιστα από έναν διαφορετικό $Rover\ B$. Επειδή οι πόροι που απαιτούνται για την επίτευξη αυτών των δύο στόχων (οι ίδιοι οι rovers) είναι disjoin (δηλαδή, δεν επικαλύπτονται), το πραγματικό κόστος για την επίτευξη και των δύο στόχων στο βέλτιστο πλάνο ($h'(n)$) πρέπει να είναι τουλάχιστον όσο το άθροισμα των (χαλαρών) κοστών τους. Ο $Rover\ A$ πρέπει να εκτελέσει το πλάνο για τον G_1 και, ταυτόχρονα, ο $Rover\ B$ πρέπει να εκτελέσει το πλάνο για τον G_2. Δεδομένου ότι τα χαλαρά κόστη είναι ήδη κατώτατα όρια για κάθε μεμονωμένο στόχο, το άθροισμά τους παραμένει ένα έγκυρο κατώτατο όριο για το συνδυαστικό πρόβλημα.

3. **Υλοποίηση σε Ψευδοκώδικα**

   Η υλοποίηση απαιτεί πρώτα τον υπολογισμό όλων των μεμονωμένων "χαλαρών" κοστών, την ταξινόμησή τους και, τέλος, την επιλογή των κατάλληλων δύο στόχων. Τα goal costs υπολογίζονται με τον ίδιο τρόπο που υπολογίζονται και στην ευρετική H2.

Algorithm 7 Ευρετική Συνάρτηση H3: Sum-2-Goals Heuristic

```
1: function CALCULATE_HEURISTIC_H3(state)
2:     all_goal_costs ← new List of GoalCost
       ▷ 1. Υπολογισμός του βέλτιστου κόστους για ΚΑΘΕ ανεκπλήρωτο στόχο
3:     Calculate_All_Individual_Goal_Costs(state, all_goal_costs)
4:     if all_goal_costs is empty then return 0
5:     end if
                    ▷ 2. Ταξινόμηση των στόχων κατά κόστος, σε φθίνουσα σειρά
6:     sort(all_goal_costs) descending by cost
                              ▷ 3. Η βάση της ευρετικής είναι ο πιο ακριβός στόχος
7:     h_value ← all_goal_costs[0].cost
8:     primary_rover_id ← all_goal_costs[0].rover_id
                    ▷ 4. Εύρεση του 2ου πιο ακριβού στόχου που εκτελείται από
   ΔΙΑΦΟΡΕΤΙΚΟ rover
9:     second_cost ← 0
10:    for i from 1 to length(all_goal_costs) - 1 do
11:        if all_goal_costs[i].rover_id ≠ primary_rover_id then
12:            second_cost ← all_goal_costs[i].cost
13:            break                    ▷ Βρέθηκε ο καλύτερος δεύτερος στόχος
14:        end if
15:    end for
                                    ▷ 5. Η τελική ευρετική είναι το άθροισμα των δύο
16:    return h_value + second_cost
17: end function
```

4.3.5 Ευρετική H4: Βέλτιστη Ανάθεση Στόχων (Optimal Assignment - Admissible)

1. **Ορισμός και Σκεπτικό**

   Η $Sum-2-Goals$ heuristic (H3) επιβεβαίωσε ότι η εκμετάλλευση της παραλληλίας των rovers οδηγεί σε μια πιο ισχυρή (πιο πληροφοριακή) ευρετική. Η **H4**, ή "**Optimal Assignment Heuristic**", αποτελεί τη λογική γενίκευση και την τελική εξέλιξη αυτής της ιδέας.

   Αντί να σταματά στους δύο πιο ακριβούς στόχους, αυτή η προσέγγιση προσπαθεί να εκμεταλλευτεί πλήρως τον διαθέσιμο παραλληλισμό. Το σκεπτικό της είναι το εξής: για να μεγιστοποιήσουμε το κατώτατο όριο (lower bound) του κόστους, πρέπει να υπολογίσουμε το μέγιστο δυνατό άθροισμα κόστους από παράλληλες ενέργειες που μπορούν να εκτελεστούν ταυτόχρονα. Για να το πετύχει αυτό, η ευρετική υλοποιεί έναν **άπληστο αλγόριθμο ανάθεσης (greedy assignment algorithm)**:

   - Υπολογίζει όλα τα πιθανά "χαλαρά" κόστη για κάθε ανεκπλήρωτο

στόχο από κάθε ικανό rover.

- Ταξινομεί όλες αυτές τις πιθανές αναθέσεις "στόχος-rover" σε φθίνουσα σειρά κόστους.
- Διατρέχει τη ταξινομημένη λίστα και αναθέτει άπληστα την πιο ακριβή διαθέσιμη "εργασία" σε κάθε ελεύθερο rover, μέχρι να απασχοληθούν όλα τα rovers.
- Το άθροισμα των κοστών αυτών των αναθέσεων, συν το απαραίτητο κόστος ενέργειας για την εκτέλεσή τους, αποτελεί την τελική τιμή της ευρετικής.

2. **Ανάλυση Παραδεκτότητας (Admissibility Analysis)**

   Παραδεκτότητα: Η ευρετική αυτή είναι **παραδεκτή (admissible)**.

   Η απόδειξη της παραδεκτότητας βασίζεται στην ίδια αρχή με την H3: την **αποσύνθεση του προβλήματος σε ανεξάρτητα υποπροβλήματα (disjoint subproblems)**. Η άπληστη ανάθεση εξασφαλίζει ότι κάθε στόχος που προστίθεται στο τελικό άθροισμα έχει ανατεθεί σε ένα **μοναδικό, διακριτό rover**. Εφόσον οι πόροι (τα rovers) που εκτελούν τις επιλεγμένες εργασίες είναι disjoin, το άθροισμα των χαλαρών (και άρα παραδεκτών) κοστών τους παραμένει ένα έγκυρο και παραδεκτό κατώτατο όριο για το συνολικό πρόβλημα.

   Επιπλέον, ο υπολογισμός του ενεργειακού κόστους είναι σχεδιασμένος ώστε να διατηρεί την παραδεκτότητα. Για κάθε rover, υπολογίζεται το ενεργειακό έλλειμμα με βάση **μόνο το κόστος της εργασίας που του έχει ανατεθεί**. Το κόστος επαναφόρτισης που προστίθεται περιλαμβάνει μόνο την απόσταση μέχρι το πλησιέστερο σημείο επαναφόρτισης, το οποίο είναι ένα αναγκαίο κόστος που πρέπει οπωσδήποτε να εκτελεστεί στο πραγματικό πλάνο.

3. **Υλοποίηση σε Ψευδοκώδικα**

   Η υλοποίηση είναι πιο σύνθετη, καθώς περιλαμβάνει τη διαχείριση και ταξινόμηση όλων των πιθανών αναθέσεων.

Algorithm 8: Ευρετική Συνάρτηση H4: Optimal Assignment Heuristic (Πλήρης)

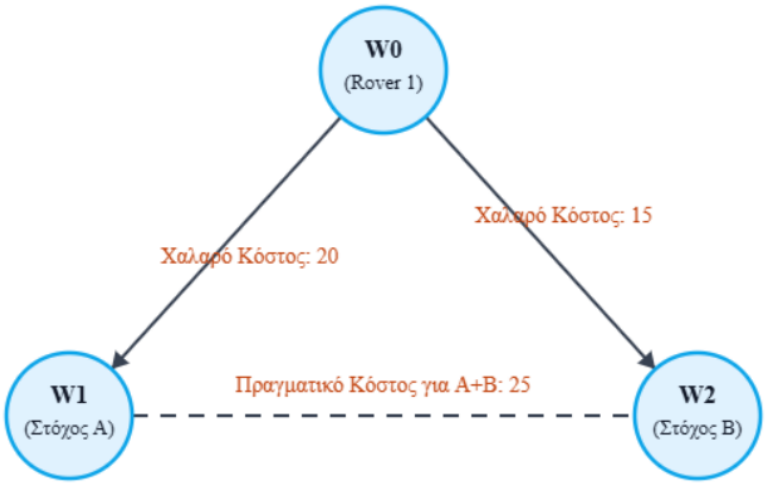
```
1: function CALCULATE_HEURISTIC_H4(state)
2:     all_assignments ← new List of GoalCost        ▷ Κάθε στοιχείο είναι (κόστος, rover_id)
3:     Calculate_All_Possible_Assignments(state, all_assignments)
4:     if all_assignments is empty then return 0
5:     end if
6:     sort(all_assignments) descending by cost
7:     h_tasks ← 0
```

```
    rover_used ← new Boolean array, initialized to FALSE
    assigned_costs ← new Integer array, initialized to 0
    for each assignment in all_assignments do
        if rover_used[assignment.rover_id] == FALSE then
            h_tasks ← h_tasks + assignment.cost
            assigned_costs[assignment.rover_id] ← assignment.cost
            rover_used[assignment.rover_id] ← TRUE
        end if
    end for
    h_energy ← Calculate_Energy_Cost_For_Assignment(state, assigned_costs)
    if h_energy == ∞ then return ∞
    end if
    return h_tasks + h_energy
end function

function CALCULATE_ALL_POSSIBLE_ASSIGNMENTS(state, assignments_list)
    for each unfulfilled_goal in state do
        for each rover r capable of achieving the goal do
            cost ← Calculate_Relaxed_Cost(goal, rover, state)
            if cost < ∞ then
                new_assignment ← (cost, rover.id)
                assignments_list.add(new_assignment)
            end if
        end for
    end for
end function

function CALCULATE_ENERGY_COST_FOR_ASSIGNMENT(state, assigned_costs)
    total_recharge_cost ← 0
    for each rover r in state do
        if assigned_costs[r] > 0 then      ▷ Αν έχει ανατεθεί εργασία στο rover
            deficit ← assigned_costs[r] - rover.energy
            if deficit > 0 then
                recharges_needed ← ⌈ deficit / 20 ⌉
                dist_to_sun ← find_min_dist_to_sunny_waypoint(rover.position)
                if dist_to_sun == ∞ then return ∞
                end if
                total_recharge_cost ← total_recharge_cost + dist_to_sun
            end if
        end if
    end for
```

47: **return** total_recharge_cost
48: **end function**

4.3.6 Πρακτική Εφαρμογή Ευρετικών σε Παράδειγμα-Παιχνίδι

Για να γίνει πιο απτή και κατανοητή η διαφορά στη συμπεριφορά των ευρετικών, ας εξετάσουμε μια απλοποιημένη, υποθετική κατάσταση, όπως αυτή που απεικονίζεται στο Σχήμα 4.2.



Το άθροισμα των χαλαρών κόστων (15 + 20 = 35) είναι παραπλανητικά μεγαλύτερο από το πραγματικό κόστος (25), καθώς δεν λαμβάνει υπόψη τις συνέργειες.

Σχήμα 4.2: Σύγκριση Χρόνου Εκτέλεσης των Μηχανισμών Εντοπισμού Διπλότυπων στο Πρόβλημα p08

Έστω ότι έχουμε **ένα μόνο rover (R1)** στην αρχική του θέση (W0) και **δύο ανεκπλήρωτους στόχους** τον A και τον B, με τα αντίστοιχα χαλαρά κόστη. Ας υποθέσουμε επίσης ότι το βέλτιστο πραγματικό πλάνο για την επίτευξη **και των δύο στόχων** έχει συνολικό κόστος $h'(n) = 25$.

Ο παρακάτω πίνακας συνοψίζει πώς κάθε ευρετική αξιολογεί την κατάσταση: Από τον πίνακα φαίνεται καθαρά η θεμελιώδης διαφορά:

- Η **H1 (Sum)** υπερεκτιμά το πραγματικό κόστος ($h1(n) > h'(n)$), καθώς "μετράει διπλά" τα κοινά μέρη του μονοπατιού. Είναι, όπως αποδείξαμε, **μη-παραδεκτή**.
- Η **H2 (Max)** υποεκτιμά το πραγματικό κόστος ($h2(n) \leq h'(n)$), καθώς αγνοεί τον δεύτερο στόχο. Είναι **παραδεκτή**, αλλά "απαισιόδοξη".

Πίνακας 4.3: Σύγκριση υπολογισμού ευρετικών στο απλοποιημένο παράδειγμα.

Ευρετική	**Υπολογισμός**	**Τιμή h(n)**	**Σχέση με $h^*(n)$**	**Ιδιότητα**
H1 (Sum)	$20 + 15$	35	$35 > 25$	Non-admissible
H2 (Max)	$\max(20, 15)$	20	$20 \leq 25$	**Admissible**

Αυτό το απλό παράδειγμα οπτικοποιεί γιατί οι πιο σύνθετες ευρετικές H3 και H4 προσπαθούν να βρουν μια χρυσή τομή, αυξάνοντας την τιμή της Max (για να είναι πιο πληροφοριακές) χωρίς να παραβιάζουν τη συνθήκη της παραδεκτότητας (όπως κάνει η Sum).

4.4 Πειραματική Διαδικασία (Experimental Setup)

Έχοντας αναλύσει την αρχιτεκτονική του planner και τον σχεδιασμό των διαφόρων ευρετικών συναρτήσεων, το επόμενο βήμα είναι η εμπειρική αξιολόγηση της απόδοσής τους. Η παρούσα ενότητα περιγράφει με λεπτομέρεια τη μεθοδολογία που ακολουθήθηκε για τη διεξαγωγή των πειραμάτων, με στόχο τη διασφάλιση της εγκυρότητας και της αναπαραγωγιμότητας των αποτελεσμάτων. Στις υποενότητες που ακολουθούν, θα οριστεί το υπολογιστικό περιβάλλον εκτέλεσης, θα παρουσιαστεί το σύνολο των προβλημάτων-αναφοράς (benchmarks) που χρησιμοποιήθηκαν, θα καθοριστούν οι μετρικές απόδοσης που καταγράφηκαν και, τέλος, θα περιγραφεί η διαδικασία επικύρωσης των παραγόμενων πλάνων.

4.4.1 Περιβάλλον εκτέλεσης

Για τη διασφάλιση της εγκυρότητας, της σταθερότητας και της αναπαραγωγιμότητας των αποτελεσμάτων, όλα τα πειράματα διεξήχθησαν σε ένα ενιαίο, αυστηρά καθορισμένο υπολογιστικό περιβάλλον. Οι μετρήσεις απόδοσης πραγματοποιήθηκαν σε απομακρυσμένο server του πανεπιστημίου με τις παρακάτω τεχνικές προδιαγραφές.

1. **Υλικό (Hardware)**:
    - **Επεξεργαστής (CPU)**: Intel(R) Core(TM) i9-10900K CPU @ 3.70GHz
    - **Μνήμη (RAM)**: 128 GB
2. **Λογισμικό (Software)**:
    - **Λειτουργικό Σύστημα**: Ubuntu 22.04.5 LTS
    - **Μεταγλωττιστής**: GCC (Ubuntu 11.4.0-1ubuntu1 22.04) 11.4.0

- **Επιλογές Μεταγλώττισης**: Η μεταγλώττιση του πηγαίου κώδικα σε C έγινε με την παρακάτω εντολή:

```
gcc -O3 -pg planner.c bloom.c -o rover_planner -lm
```

Είναι κρίσιμο να σημειωθεί η χρήση της σημαίας $-O3$, η οποία ενεργοποιεί το υψηλότερο επίπεδο βελτιστοποίησης του μεταγλωττιστή, διασφαλίζοντας ότι οι μετρήσεις χρόνου αντικατοπτρίζουν τη μέγιστη δυνατή απόδοση του υλοποιημένου κώδικα. Η σημαία $-pg$ χρησιμοποιήθηκε για τη δημιουργία προφίλ εκτέλεσης (profiling), επιτρέποντας την ανάλυση του χρόνου που καταναλώνεται σε κάθε συνάρτηση.

4.4.2 Σύνολο Δεδομένων (Benchmarks)

Για την αξιολόγηση του planner επιλέχθηκε το επίσημο σύνολο προβλημάτων-αναφοράς (benchmarks) από το αποθετήριο του **Numeric Track του IPC 2023**. Συγκεκριμένα, χρησιμοποιήθηκαν όλες οι **20 διαθέσιμες περιπτώσεις (instances)** του πεδίου 'Rover'[4], οι οποίες κυμαίνονται από το $pfile1.pddl$ έως το $pfile20.pddl$.

Η επιλογή του συγκεκριμένου συνόλου δεδομένων είναι θεμελιώδους σημασίας για την επίτευξη του κεντρικού στόχου της παρούσας εργασίας. Η χρήση των ίδιων ακριβώς προβλημάτων που χρησιμοποιήθηκαν για την αξιολόγηση των state-of-the-art domain-independent planners στον διαγωνισμό, επιτρέπει μια **άμεση, αντικειμενική και δίκαιη σύγκριση** των αποτελεσμάτων. Με αυτόν τον τρόπο, η απόδοση του εξειδικευμένου μας planner μπορεί να μετρηθεί απευθείας έναντι της απόδοσης των κορυφαίων συστημάτων γενικής χρήσης, στο ίδιο ακριβώς πεδίο δοκιμών. Οι περιπτώσεις αυτές παρουσιάζουν κλιμακούμενη δυσκολία, περιλαμβάνοντας διαφορετικό αριθμό οχημάτων, στόχων και τοποθεσιών, επιτρέποντας έτσι μια σφαιρική αξιολόγηση της απόδοσης και της επεκτασιμότητας (scalability) του planner.

4.4.3 Μετρικές Απόδοσης

Για την ποσοτική και σφαιρική αξιολόγηση της απόδοσης του planner και των ευρετικών συναρτήσεων, καταγράφηκε μια σειρά από καθιερωμένες μετρικές. Οι μετρικές αυτές επιτρέπουν τη σύγκριση τόσο μεταξύ των διαφορετικών εκδοχών του υλοποιημένου planner, όσο και με άλλα συστήματα σχεδιασμού.

- **Ποιότητα Πλάνου (Plan Cost)**: Ως κύρια μετρική για την ποιότητα της λύσης ορίζεται η τελική τιμή της συνάρτησης $(recharges)$. Δεδομένου ότι

[4]Μπορείτε να δείτε αναλυτικά όλες τις περιγραφές προβλημάτων στον σύνδεσμο: https://github.com/ipc2023-numeric/ipc2023-dataset/tree/main/rover/instances

ο στόχος βελτιστοποίησης στα προβλήματα του 'Rover' είναι η ελαχιστοποίηση των επαναφορτίσεων, ένα πλάνο θεωρείται υψηλότερης ποιότητας όσο μικρότερος είναι ο αριθμός αυτός.

- **Χρόνος Επίλυσης (CPU Time)**: Ο χρόνος που απαιτείται για την εύρεση μιας λύσης. Η μέτρηση αφορά τον **χρόνο CPU** που καταναλώνει η διεργασία του planner, και όχι τον συνολικό χρόνο που παρέρχεται (wall-clock time). Ο χρόνος μετρήθηκε σε δευτερόλεπτα, χρησιμοποιώντας τη συνάρτηση $clock()$ της τυπικής βιβλιοθήκης της C.

  Είναι σημαντικό να σημειωθεί ότι, ενώ η μέτρηση της τελικής απόδοσης βασίζεται στον χρόνο CPU, ο μηχανισμός τερματισμού (timeout) των 10 λεπτών υλοποιήθηκε με βάση τον πραγματικό χρόνο (wall-clock time) μέσω της συνάρτησης $time()$. Για την ελαχιστοποίηση της επίδρασης από πιθανές καθυστερήσεις σε ένα περιβάλλον με μεταβαλλόμενο φόρτο, η πλειοψηφία των πειραμάτων διεξήχθη σε ώρες μειωμένης κίνησης.

- **Κόπος Αναζήτησης (Search Effort)**: Για την εκτίμηση του υπολογιστικού κόπου της αναζήτησης, ανεξάρτητα από την ταχύτητα του επεξεργαστή, καταγράφηκαν δύο μετρήσεις:

  1. **Κόμβοι που Δημιουργήθηκαν (Generated Nodes)**: Το συνολικό πλήθος των κόμβων που δημιουργήθηκαν και εισήχθησαν στο Frontier.
  2. **Κόμβοι που Επεκτάθηκαν (Expanded Nodes)**: Το συνολικό πλήθος των κόμβων που εξήχθησαν από το Frontier για να εξερευνηθούν οι διάδοχοί τους.

- **Χρήση Μνήμης (Memory Usage)**: Δεν πραγματοποιήθηκε άμεση μέτρηση της μέγιστης χρήσης μνήμης RAM. Ωστόσο, η χρήση μνήμης είναι ευθέως ανάλογη με το μέγιστο μέγεθος των λιστών OPEN και CLOSED. Επομένως, ο αριθμός των κόμβων που δημιουργήθηκαν και αποθηκεύτηκαν χρησιμοποιείται ως ένας καλός δείκτης (proxy) για την εκτίμηση των απαιτήσεων σε μνήμη.

Τέλος, είναι αξιοσημείωτο ότι στα αρχεία λύσεων καταγράφονται και οι τιμές f και h του τελικού μονοπατιού. Αν και δεν αποτελούν πρωταρχικές μετρικές απόδοσης, χρησιμοποιήθηκαν κατά τη φάση της ανάλυσης για τη διαγνωστική μελέτη της συμπεριφοράς και την εμπειρική επιβεβαίωση της παραδεκτότητας της κάθε ευρετικής.

4.4.4 Επικύρωση Λύσεων

Για τη διασφάλιση της ορθότητας και της εγκυρότητας των πειραματικών αποτελεσμάτων, αναπτύχθηκε ένα εξειδικευμένο εργαλείο επικύρωσης, το $rover\_verify.c$.

Ο σκοπός αυτού του εργαλείου είναι να επιβεβαιώσει ότι κάθε πλάνο που παράγεται από τον planner είναι όντως μια έγκυρη λύση στο αντίστοιχο πρόβλημα, επιβεβαιώνοντας έτσι την ορθή λειτουργία του planner.

Η διαδικασία επικύρωσης είναι η εξής: το εργαλείο δέχεται ως ορίσματα γραμμής εντολών το αρχείο του προβλήματος και το αρχείο της λύσης που παρήγαγε ο planner ($rover\_verify < problem - file > < solution - file >$). Αρχικά, αναλύει το αρχείο του προβλήματος για να ανακατασκευάσει την αρχική κατάσταση. Στη συνέχεια, διαβάζει το αρχείο της λύσης, αναλύοντας κάθε δράση του πλάνου με τη σειρά. Για κάθε δράση, ο επικυρωτής προσομοιώνει την εκτέλεσή της:

1. **Ελέγχει αν οι προϋποθέσεις** (τόσο οι λογικές όσο και οι αριθμητικές) της δράσης ικανοποιούνται στην τρέχουσα κατάσταση.

2. Αν οι προϋποθέσεις είναι έγκυρες, **εφαρμόζει τα αποτελέσματα** της δράσης για να δημιουργήσει την επόμενη κατάσταση.

Αν σε οποιοδήποτε βήμα μια προϋπόθεση δεν ικανοποιείται, ο επικυρωτής τερματίζει και αναφέρει το πλάνο ως άκυρο. Εάν ολόκληρη η ακολουθία των δράσεων εκτελεστεί επιτυχώς, πραγματοποιείται ένας τελικός έλεγχος για να διαπιστωθεί εάν η τελική κατάσταση που προέκυψε ικανοποιεί όλες τις συνθήκες-στόχο του προβλήματος. Μόνο αν και αυτός ο έλεγχος είναι επιτυχής, το πλάνο χαρακτηρίζεται ως έγκυρο.

4.5 Πειραματικά Αποτελέσματα και Ανάλυση

Αφού περιγράφηκε η μεθοδολογία υλοποίησης του planner και η πειραματική διαδικασία, η παρούσα ενότητα παρουσιάζει και αναλύει τα δεδομένα που συλλέχθηκαν. Αυτή η ενότητα αποτελεί τον πυρήνα της εμπειρικής αξιολόγησης, όπου οι θεωρητικές ιδιότητες και το σκεπτικό πίσω από τον σχεδιασμό των ευρετικών συναρτήσεων δοκιμάζονται στην πράξη. Η ανάλυση που ακολουθεί είναι δομημένη γύρω από τρεις βασικούς άξονες: αρχικά, θα παρουσιαστούν τα συγκεντρωτικά αποτελέσματα για όλες τις ευρετικές και τα προβλήματα-αναφοράς. Στη συνέχεια, θα γίνει μια σε βάθος ερμηνεία αυτών των αποτελεσμάτων, συγκρίνοντας την απόδοση των ευρετικών και αξιολογώντας εμπειρικά τη σημασία της παραδεκτότητας. Τέλος, θα εξεταστεί η επίδραση του υβριδικού μηχανισμού εντοπισμού διπλότυπων στην απόδοση του planner.

4.5.1 Συγκεντρωτικά Αποτελέσματα Απόδοσης

Τα πρωτογενή δεδομένα που συλλέχθηκαν από την εκτέλεση των πειραμάτων παρουσιάζονται στους παρακάτω συγκεντρωτικούς πίνακες. Είναι σημαντικό να σημειωθεί ότι δεν εκτελέστηκαν όλοι οι 10 πιθανοί συνδυασμοί αλγορίθμου-ευρετικής. Αντιθέτως, επιλέχθηκε ένα υποσύνολο 6 συνδυασμών, οι οποίοι έχουν

το μεγαλύτερο επιστημονικό ενδιαφέρον και απαντούν πιο άμεσα στα ερευνητικά ερωτήματα της εργασίας.

Η επιλογή έγινε με βάση τον σκοπό του κάθε αλγορίθμου:

- Ο **Αλγόριθμος A***, ως αλγόριθμος βέλτιστης σχεδίασης, συνδυάστηκε αποκλειστικά με τις **παραδεκτές (admissible) ευρετικές (H0, H2, H3, H4)**, καθώς μόνο αυτοί οι συνδυασμοί εγγυώνται την εύρεση της βέλτιστης λύσης.

- Ο **Αλγόριθμος Πρώτα-στο-Καλύτερο (GBFS)**, ως αλγόριθμος ικανοποιητικής σχεδίασης, συνδυάστηκε με την κατεξοχήν "άπληστη" μη-παραδεκτή ευρετική **(H1)**, καθώς και με την πιο πληροφοριακή παραδεκτή ευρετική **(H4)**, για να μελετηθεί η συμπεριφορά της σε ένα καθαρά greedy πλαίσιο.

Αυτή η στοχευμένη επιλογή μάς επιτρέπει να κάνουμε τις πιο ουσιώδεις συγκρίσεις. Στους πίνακες που ακολουθούν, παρουσιάζονται οι βασικές μετρικές απόδοσης. Οι περιπτώσεις όπου δεν βρέθηκε λύση, είτε λόγω υπέρβασης του **χρονικού ορίου των 600 δευτερολέπτων (10 λεπτών)** είτε λόγω **εξάντλησης της διαθέσιμης μνήμης**, σημειώνονται με '-'. Το χρονικό αυτό όριο επιλέχθηκε καθώς αποτελεί ένα συνηθισμένο πρότυπο σε ακαδημαϊκούς διαγωνισμούς σχεδιασμού, όπως ο IPC, προσφέροντας μια δίκαιη ισορροπία μεταξύ της παροχής επαρκούς χρόνου για την επίλυση απαιτητικών προβλημάτων και της διατήρησης της συνολικής διάρκειας των πειραμάτων σε λογικά πλαίσια.

4.5.2 Οπτικοποίηση των Αποτελεσμάτων

Για την καλύτερη κατανόηση των τάσεων που υποκρύπτονται στους πίνακες των δεδομένων, ακολουθεί (μετά τους πίνακες) μια σειρά από γραφήματα που οπτικοποιούν τις βασικές πτυχές της απόδοσης των planners. Κάθε γράφημα συνοδεύεται από ένα σύντομο σχόλιο που επισημαίνει το κύριο συμπέρασμα που μπορεί να εξαχθεί.

Το ραβδόγραμμα του Σχήματος 4.3 συνοψίζει την **κάλυψη (coverage)** κάθε διαμόρφωσης, δηλαδή το πλήθος των προβλημάτων (από τα 20) που κατάφερε να επιλύσει εντός του χρονικού ορίου. Είναι σαφές ότι η μη-παραδεκτή $H1 + Best - First$ λύνει τα περισσότερα προβλήματα (9/20), ενώ η $H4 + A*$ αποτελεί τον ισχυρότερο βέλτιστο planner (8/20). Η απόλυτη αποτυχία της $H0 + A*$ (4/20) καταδεικνύει την ανάγκη για ευρετική καθοδήγηση.

Το γράφημα γραμμών του Σχήματος 4.4 απεικονίζει τον κόπο αναζήτησης (κόμβοι που επεκτάθηκαν) για τους βέλτιστους planners σε λογαριθμική κλίμακα. Η διαφορά απόδοσης είναι δραματική: η $H0$ εξερευνά εκατομμύρια κόμβους, ενώ οι πληροφοριακές ευρετικές $H2$, $H3$, και $H4$ μειώνουν τον χώρο αναζήτησης κατά πολλές τάξεις μεγέθους. Φαίνεται καθαρά η "σκάλα" βελτίωσης της καθοδήγησης, με τις H3 και H4 να είναι οι πιο αποδοτικές.

Πίνακας 4.4: Συνολικός Αριθμός Βημάτων (Plan Steps) για κάθε Λύση.

Problem	H0+A*	H1+best-first	H2+A*	H3+A*	H4+A*	H4+best-first
p01	8	8	8	8	8	8
p02	10	10	10	10	10	10
p03	12	14	11	11	11	15
p04	9	8	9	8	9	9
p05	-	24	-	-	-	24
p06	-	-	-	-	-	-
p07	-	18	-	-	20	-
p08	-	26	-	-	27	-
p09	-	-	-	-	-	-
p10	-	-	-	-	-	-
p11	-	20	-	-	20	-
p12	-	-	-	-	-	-
p13	-	-	-	-	-	-
p14	-	-	-	-	-	-
p15	-	-	-	-	-	-
p16	-	-	-	-	-	-
p17	-	-	-	-	-	-
p18	-	50	-	-	-	-
p19	-	-	-	-	-	-
p20	-	-	-	-	-	-

Το διάγραμμα διασποράς του Σχήματος 4.5 οπτικοποιεί τον **συμβιβασμό ταχύτητας-ποιότητας** για τις δύο ικανοποιητικές (satisficing) διαμορφώσεις. Η H1+Best-First βρίσκει λύσεις σχεδόν ακαριαία (πολύ κοντά στον άξονα y), αλλά είναι η μόνη που παράγει πλάνα με κόστος επαναφορτίσεων > 0 (υψηλότερα στον άξονα y). Η H4+Best-First, αν και ελαφρώς πιο αργή, καταφέρνει να διατηρήσει τη βέλτιστη ποιότητα πλάνου (κόστος 0) στα προβλήματα που επιλύει.

4.5.3 Ανάλυση της Απόδοσης των Ευρετικών

Η συνδυαστική μελέτη των πινάκων και των γραφημάτων της προηγούμενης ενότητας επιτρέπει την εξαγωγή μιας σειράς από ουσιαστικά συμπεράσματα, τα οποία όχι μόνο απαντούν στα αρχικά ερευνητικά ερωτήματα, αλλά αποκαλύπτουν και τις λεπτές αποχρώσεις της συμπεριφοράς του κάθε συνδυασμού αλγορίθμου-ευρετικής.

1. **Η Θεμελιώδης Ανάγκη για Ευρετική Καθοδήγηση**

   Η απόδοση του συνδυασμού **H0 + A***, που ισοδυναμεί με τον αλγόριθμο Dijkstra, είναι αποκαλυπτική. Όπως φαίνεται στο Σχήμα 4.5, η συγκεκρι-

Πίνακας 4.5: Κόστος Πλάνου (Αριθμός Επαναφορτίσεων) για κάθε Συνδυασμό.

Problem	H0+A*	H1+best-first	H2+A*	H3+A*	H4+A*	H4+best-first
p01	0	0	0	0	0	0
p02	0	0	0	0	0	0
p03	0	0	0	0	0	0
p04	0	0	0	0	0	0
p05	-	0	-	-	-	0
p06	-	-	-	-	-	-
p07	-	0	-	-	0	-
p08	-	0	-	-	0	-
p09	-	-	-	-	-	-
p10	-	-	-	-	-	-
p11	-	0	-	-	0	-
p12	-	-	-	-	-	-
p13	-	-	-	-	-	-
p14	-	-	-	-	-	-
p15	-	-	-	-	-	-
p16	-	-	-	-	-	-
p17	-	-	-	-	-	-
p18	-	2	-	-	-	-
p19	-	-	-	-	-	-
p20	-	-	-	-	-	-

μένη ρύθμιση καταφέρνει να επιλύσει μόνο τα 4 απλούστερα προβλήματα. Η αποτυχία της σε οτιδήποτε πιο σύνθετο, λόγω της εκρηκτικής αύξησης του χώρου αναζήτησης (Πίνακας 4.6, 4.7), αποδεικνύει εμπειρικά ότι μια "τυφλή" αναζήτηση είναι υπολογιστικά ασύμφορη για το πρόβλημα 'Rover'. Το αποτέλεσμα αυτό, αν και αναμενόμενο, είναι κρίσιμο, καθώς θεμελιώνει την απόλυτη ανάγκη για την ανάπτυξη πληροφοριακών ευρετικών συναρτήσεων.

2. **Ο Συμβιβασμός Ταχύτητας-Ποιότητας: Η "Άπληστη" Προσέγγιση**

   Ο συνδυασμός του **best-first-search με τη μη-παραδεκτή ευρετική H1 (Sum)** αναδεικνύεται ως ο ταχύτερος ικανοποιητικός (satisficing) planner, έχοντας την υψηλότερη κάλυψη (9/20 προβλήματα). Ωστόσο, αυτή η ταχύτητα έχει τίμημα στην ποιότητα. Είναι ο μόνος συνδυασμός που παράγει **υποβέλτιστα πλάνα** σε προβλήματα όπως το p18 (2 recharges), επιβεβαιώνοντας τη θεωρία.

   Ακόμα πιο ενδιαφέρουσα είναι η παρατήρηση ότι η H1+GBFS βρίσκει λύσεις με **λιγότερα βήματα** από τις βέλτιστες (π.χ., 8 βήματα στο $p04$ έναντι 9 της H4+A*), παρότι και οι δύο έχουν το ίδιο, βέλτιστο κόστος (0 recharges, 33 energy). Αυτό δεν είναι παράδοξο, αλλά απόδειξη της ύπαρξης **πολ-**

Πίνακας 4.6: Συνολικό Κόστος Ενέργειας (g-score) για κάθε Λύση.

Problem	H0+A*	H1+best-first	H2+A*	H3+A*	H4+A*	H4+best-first
p01	25	25	25	25	25	25
p02	41	41	41	41	41	41
p03	49	73	49	49	49	77
p04	33	33	33	33	33	41
p05	-	86	-	-	-	85
p06	-	-	-	-	-	-
p07	-	74	-	-	74	-
p08	-	106	-	-	106	-
p09	-	-	-	-	-	-
p10	-	-	-	-	-	-
p11	-	92	-	-	84	-
p12	-	-	-	-	-	-
p13	-	-	-	-	-	-
p14	-	-	-	-	-	-
p15	-	-	-	-	-	-
p16	-	-	-	-	-	-
p17	-	-	-	-	-	-
p18	-	243	-	-	-	-
p19	-	-	-	-	-	-
p20	-	-	-	-	-	-

λαπλών, εξίσου βέλτιστων λύσεων. Ο A* εγγυάται την εύρεση μιας από αυτές, ενώ ο "άπληστος" GBFS, ακολουθώντας έναν διαφορετικό δρόμο, έτυχε να βρει μια άλλη, ισοδύναμη λύση που ήταν πιο σύντομη σε βήματα, χωρίς αυτό να την καθιστά ανώτερη ως προς τα κριτήρια βελτιστοποίησης που έχουμε θέσει.

3. **Η Εξέλιξη των Παραδεκτών Ευρετικών και το "Παράδοξο" της Πιο Έξυπνης**

   Η σύγκριση των παραδεκτών ευρετικών με τον A* αποκαλύπτει μια σαφή πορεία βελτίωσης, αλλά και μια σημαντική πολυπλοκότητα.

   - **H2 (Max) και H3 (Sum-2)**: Αν και μειώνουν δραστικά τον κόπο αναζήτησης σε σχέση με την H0, δεν καταφέρνουν να αυξήσουν τον αριθμό των επιλυμένων προβλημάτων. Αυτό υποδηλώνει ότι η καθοδήγησή τους, αν και βελτιωμένη, παραμένει ανεπαρκής για τα πιο δύσκολα σενάρια.
   - **H4 (Optimal Assignment)**: Είναι ο αδιαμφισβήτητος "νικητής" των βέλτιστων planners. Είναι η **μόνη παραδεκτή ευρετική που λύνει τα δύσκολα προβλήματα** $p07$, $p08$ και $p11$, αποδεικνύοντας ότι η

Πίνακας 4.7: Χρόνος Επίλυσης (CPU Time σε s) για κάθε Συνδυασμό.

Problem	**H0+A***	**H1+best-first**	**H2+A***	**H3+A***	**H4+A***	**H4+best-first**
p01	0.057	0.005	0.039	0.041	0.050	0.018
p02	0.035	0.0004	0.021	0.022	0.025	0.018
p03	0.652	0.001	0.076	0.061	0.213	0.019
p04	0.454	0.002	0.096	0.036	0.029	0.002
p05	-	0.007	-	-	-	0.061
p06	-	-	-	-	-	-
p07	-	0.004	-	-	0.362	-
p08	-	0.007	-	-	51.348	-
p09	-	-	-	-	-	-
p10	-	-	-	-	-	-
p11	-	0.008	-	-	30.473	-
p12	-	-	-	-	-	-
p13	-	-	-	-	-	-
p14	-	-	-	-	-	-
p15	-	-	-	-	-	-
p16	-	-	-	-	-	-
p17	-	-	-	-	-	-
p18	-	0.413	-	-	-	-
p19	-	-	-	-	-	-
p20	-	-	-	-	-	-

πιο εξελιγμένη λογική της (η άπληστη ανάθεση που εκμεταλλεύεται την παραλληλία) είναι αυτή που παρέχει την απαραίτητη ισχύ για να καθοδηγήσει την αναζήτηση αποτελεσματικά.

Ωστόσο, η H4 δεν είναι πάντα η πιο αποδοτική σε αριθμό κόμβων. Στο πρόβλημα $p08$, η H3 χρειάζεται σημαντικά λιγότερους κόμβους από την H4 (24k vs 106k). Αυτό αναδεικνύει τον κλασικό συμβιβασμό μεταξύ **πληροφοριακότητας** και **υπολογιστικού κόστους** μιας ευρετικής. Η H4 είναι πιο "έξυπνη", αλλά ο υπολογισμός της είναι πιο "ακριβός" σε χρόνο CPU, με αποτέλεσμα σε ορισμένες περιπτώσεις μια απλούστερη ευρετική να είναι συνολικά ταχύτερη.

Τέλος, η σύγκριση της ίδιας ευρετικής (H4) με τους δύο αλγορίθμους είναι αποκαλυπτική. Η **H4+GBFS** είναι ταχύτατη στα εύκολα προβλήματα, αλλά αποτυγχάνει στα πιο σύνθετα, όπου η πιο μεθοδική και ισορροπημένη προσέγγιση του A* (που ζυγίζει το g και το h) είναι απαραίτητη για να βρεθεί η λύση.

Πίνακας 4.8: Κόμβοι που Δημιουργήθηκαν (Generated Nodes) για κάθε Συνδυασμό.

Problem	H0+A*	H1+best-first	H2+A*	H3+A*	H4+A*	H4+best-first
p01	6861	59	2704	2704	2704	116
p02	4030	42	1319	1319	1319	337
p03	75849	103	5134	4071	4627	164
p04	52964	75	6094	2337	1753	147
p05	7267563	268	1049690	-	1267172	1545
p06	7148594	2339756	-	1669856	918785	1751547
p07	7223874	299	1474654	1273017	24986	-
p08	7263940	552	2602593	261451	1190457	1937768
p09	7415931	1836909	404650	1528392	900032	961354
p10	7321684	1529278	1409225	1006273	-	1642863
p11	7364871	530	-	301229	764053	997143
p12	7311327	1959742	1436438	1488969	1074209	1212859
p13	7330737	706762	2472612	413049	133544	1019919
p14	7293219	1596528	546396	875314	910312	-
p15	3886409	1141068	1367134	471241	1360007	803077
p16	3815818	-	1031139	2762280	1923853	-
p17	3973883	150118	1002558	2650877	-	2627912
p18	7331085	3110	-	880478	1577765	2008217
p19	7408996	1086533	2264508	1693138	549694	2136071
p20	3949136	1521488	-	576808	1620115	965770

4.5.4 Εμπειρική Αξιολόγηση της Παραδεκτότητας

Ένα από τα κεντρικά ερωτήματα της παρούσας εργασίας είναι η εμπειρική διερεύνηση της σημασίας της παραδεκτότητας (admissibility). Η σύγκριση της απόδοσης της μη-παραδεκτής $H1(Sum)$ με τις παραδεκτές ευρετικές παρέχει μια σαφή και ποσοτικοποιήσιμη εικόνα.

Η περίπτωση του προβλήματος $p18$ είναι η πιο αποκαλυπτική. Είναι **η μοναδική περίπτωση όπου ένας συνδυασμός, ο $H1 + Best - First$, καταφέρνει να βρει λύση, ενώ όλοι οι βέλτιστοι planners (που χρησιμοποιούν παραδεκτές ευρετικές) αποτυγχάνουν**. Αυτό αναδεικνύει την ισχύ της "άπληστης" καθοδήγησης της H1, η οποία μπορεί να "σπάσει" τα όρια πολυπλοκότητας που σταματούν τις πιο συντηρητικές, παραδεκτές ευρετικές.

Ωστόσο, η λύση που βρίσκει η H1 για το p18 έχει κόστος **2 επαναφορτίσεις**. Παρόλο που δεν μπορούμε να ισχυριστούμε με βεβαιότητα ότι η λύση αυτή είναι υποβέλτιστη (καθώς δεν διαθέτουμε βέλτιστη λύση για σύγκριση), το μη μηδενικό κόστος αποτελεί ισχυρή ένδειξη της φύσης της. Η H1, ως μη-παραδεκτή, δεν παρέχει καμία εγγύηση για την ποιότητα της λύσης. Το γεγονός ότι βρίσκει μια λύση με κόστος, ενώ σε όλα τα άλλα επιλυμένα προβλήματα το βέλτιστο κόστος

Πίνακας 4.9: Κόμβοι που Επεκτάθηκαν (Expanded Nodes) για κάθε Συνδυασμό.

Problem	H0+A*	H1+best-first	H2+A*	H3+A*	H4+A*	H4+best-first
p01	2871	10	547	547	547	18
p02	2499	12	463	463	463	103
p03	32061	19	1120	834	1103	32
p04	18250	12	1263	472	357	22
p05	1808419	39	162105	-	187071	159
p06	3761420	279006	-	426243	171011	183755
p07	1425290	26	203078	167565	2487	-
p08	873961	45	266478	24063	106780	474747
p09	644191	301122	29088	109325	44314	59540
p10	583400	129068	96398	69877	-	164283
p11	1002813	48	-	29605	76448	176852
p12	762892	212486	110529	110304	97644	80102
p13	529315	39067	168671	29565	7087	122898
p14	429171	357155	28247	52884	45487	-
p15	328390	118889	72994	23009	64455	43194
p16	340449	-	65717	178253	148852	-
p17	188532	9707	43432	115897	-	130401
p18	205282	103	-	23488	43597	56532
p19	187671	55077	55562	41604	14721	113542
p20	78509	49618	-	11493	30881	34789

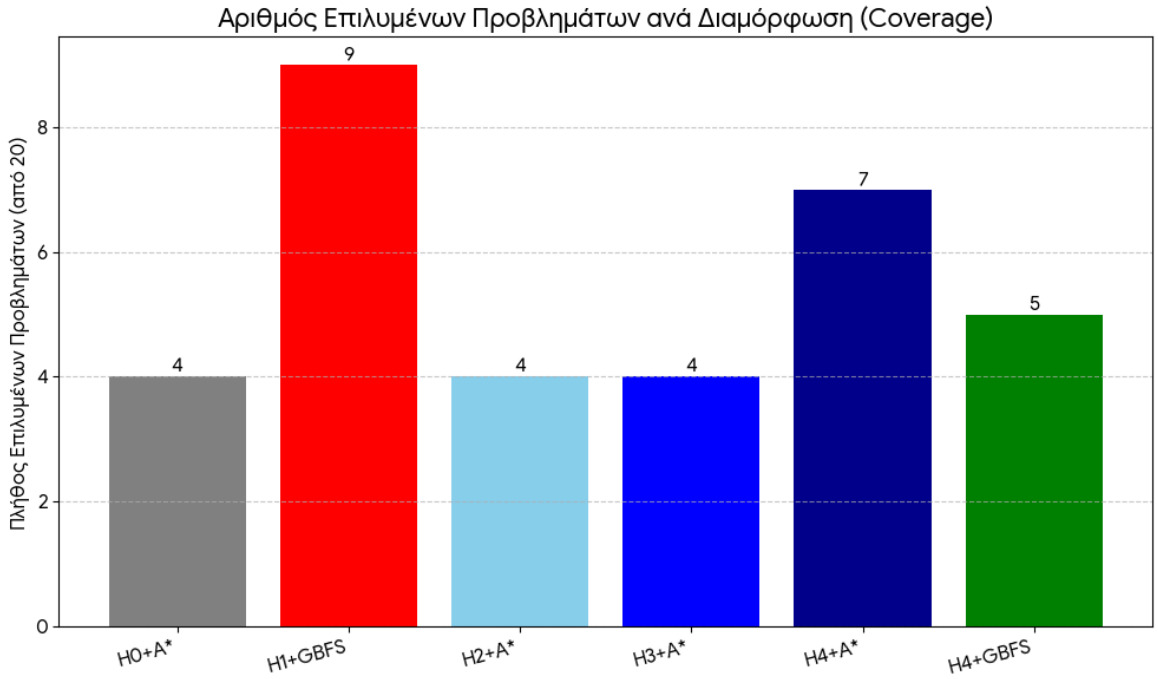
είναι 0, επιβεβαιώνει εμπειρικά τον θεωρητικό της περιορισμό.

Συμπερασματικά, η H1 αποδεικνύεται μια εξαιρετικά αποτελεσματική **satisficing** ευρετική. Η επιθετική της καθοδήγηση της επιτρέπει να λύνει προβλήματα που οι παραδεκτές ευρετικές δεν μπορούν, αλλά αυτό γίνεται με σαφές ρίσκο ως προς την ποιότητα του τελικού πλάνου.

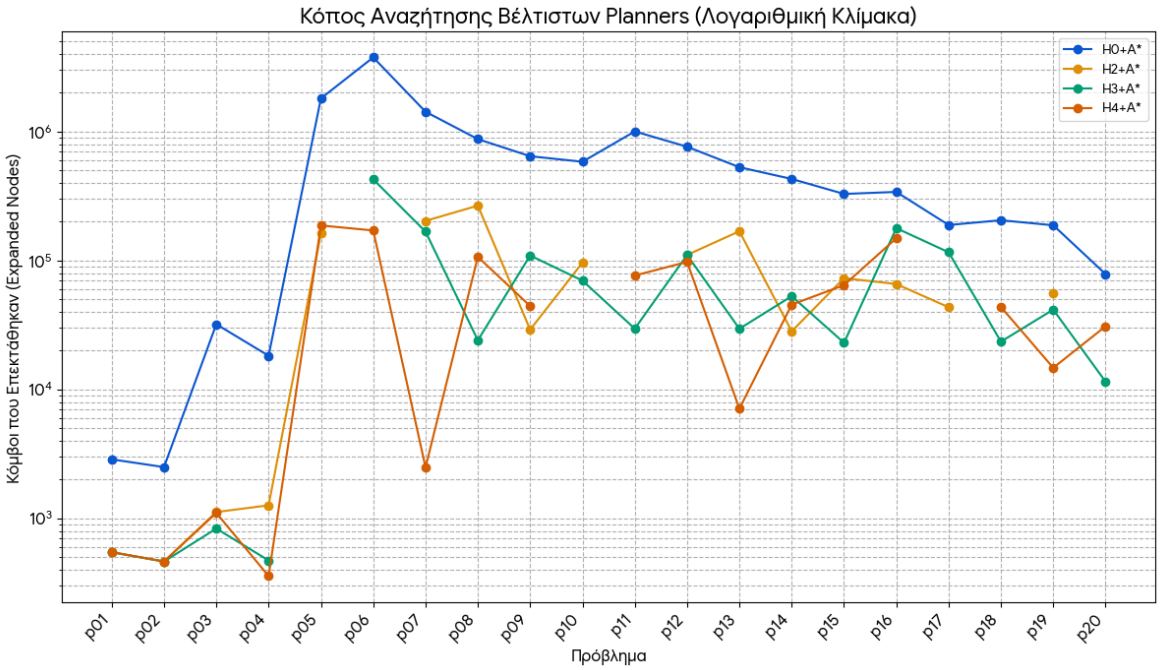
4.5.5 Ποιοτική Ανάλυση Παραγόμενων Πλάνων: Η Περίπτωση του $p04$

Πέρα από την ποσοτική σύγκριση, η ποιοτική ανάλυση των ίδιων των πλάνων που παράγονται μπορεί να αποκαλύψει σημαντικές λεπτομέρειες για τη συμπεριφορά των αλγορίθμων. Η περίπτωση του προβλήματος $p04$ είναι ιδανική για μια τέτοια μελέτη, καθώς τόσο η βέλτιστη διαμόρφωση ($H4 + A*$) όσο και η ικανοποιητική ($H1 + Best - First$) βρίσκουν λύσεις με το **ίδιο, βέλτιστο κόστος** (0 recharges, 33 energy), αλλά με **διαφορετικό αριθμό βημάτων** (9 έναντι 8).

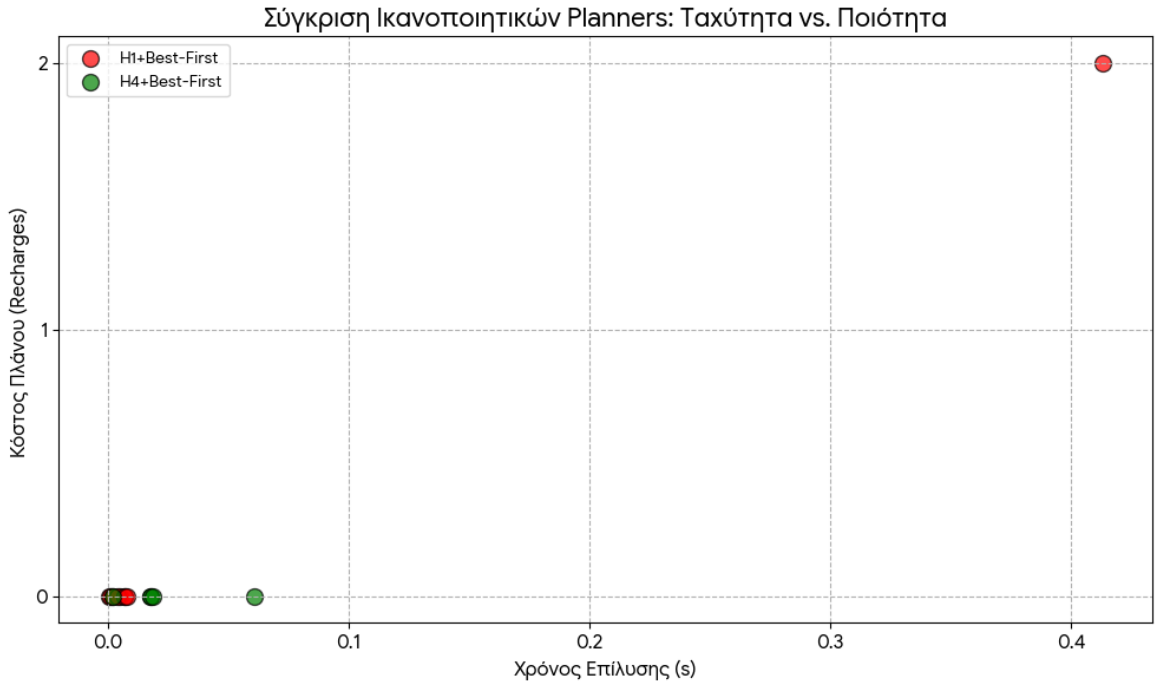
Ο πίνακας 4.10 παραθέτει τις δύο ακολουθίες δράσεων δίπλα-δίπλα για άμεση σύγκριση.



Σχήμα 4.3: Αριθμός Επιλυμένων Προβλημάτων ανά Διαμόρφωση (Coverage).



Σχήμα 4.4: Κόπος Αναζήτησης Βέλτιστων Planners (Λογαριθμική Κλίμακα).



Σχήμα 4.5: Σύγκριση Ικανοποιητικών Planners: Ταχύτητα vs. Ποιότητα.

Ερμηνεία

Η βασική διαφορά εντοπίζεται στο **4ο βήμα** του πλάνου της H4+A*: η δράση ($drop\ rover0\ store0$). Το rover0, αφού έχει ολοκληρώσει την αποστολή του (δειγματοληψία και επικοινωνία), αδειάζει τον αποθηκευτικό του χώρο. Αυτή η δράση, αν και λογική, είναι περιττή για την επίτευξη των στόχων του προβλήματος.

Αυτό το εύρημα αναδεικνύει μια κρίσιμη διαφορά στη φιλοσοφία των δύο αλγορίθμων:

- Ο **A***, με την παραδεκτή ευρετική H4, εξερευνά συστηματικά τον χώρο αναζήτησης για να βρει ένα μονοπάτι που **εγγυημένα** έχει το ελάχιστο κόστος. Στη διαδικασία αυτή, βρήκε το μονοπάτι των 9 βημάτων. Επειδή η δράση $drop$ έχει μηδενικό κόστος ενέργειας και δεν προσθέτει επαναφορτίσεις, το πλάνο αυτό είναι **απολύτως βέλτιστο** σύμφωνα με τα κριτήριά μας.

- Ο **Best-First Search**, με την "άπληστη" ευρετική H1, ακολούθησε έναν πιο ευκαιριακό δρόμο. Έτυχε να βρει ένα διαφορετικό μονοπάτι που επίσης ικανοποιεί όλους τους στόχους με το ίδιο βέλτιστο κόστος, παραλείποντας την περιττή δράση $drop$.

Συμπερασματικά, η ανάλυση αυτή αποδεικνύει έμπρακτα την ύπαρξη **πολλαπλών, εξίσου βέλτιστων λύσεων**. Το γεγονός ότι ο ικανοποιητικός planner βρήκε μια λύση με λιγότερα βήματα δεν τον καθιστά ανώτερο, αλλά απλώς

Πίνακας 4.10: Σύγκριση των παραγόμενων πλάνων για το πρόβλημα p04.

Βήμα	**Πλάνο από H4+A* (9 βήματα)**	**Πλάνο από H1+Best-First (8 βήματα)**
1	(sample_soil rover0 store0 waypoint3)	(navigate rover1 waypoint2 waypoint1)
2	(communicate_soil_data ...)	(sample_rock rover1 store1 waypoint1)
3	(navigate rover1 waypoint2 waypoint1)	(communicate_rock_data ...)
4	**(drop rover0 store0)**	(sample_soil rover0 store0 waypoint3)
5	(calibrate rover1 camera0 ...)	(communicate_soil_data ...)
6	(take_image rover1 ...)	(calibrate rover1 camera0 ...)
7	(sample_rock rover1 store1 waypoint1)	(take_image rover1 ...)
8	(communicate_rock_data ...)	(communicate_image_data ...)
9	(communicate_image_data ...)	-

αναδεικνύει ότι η ”βελτιστότητα” εξαρτάται αυστηρά από τη μετρική που έχει οριστεί.

4.5.6 Ανάλυση του Μηχανισμού Εντοπισμού Διπλότυπων

Για την εμπειρική αξιολόγηση του υβριδικού μηχανισμού εντοπισμού διπλότυπων, διεξήχθη ένα στοχευμένο πείραμα. Η καλύτερη διαμόρφωση του planner (H4+A*) εκτελέστηκε σε ένα αντιπροσωπευτικό υποσύνολο προβλημάτων με τρεις διαφορετικές ρυθμίσεις: χρησιμοποιώντας μόνο το **Hash Set (’Hash only”)**, μόνο το **Bloom Filter (’Bloom only”)**, και τον **υβριδικό μηχανισμό** που συνδυάζει και τα δύο.

Τα αποτελέσματα, που συνοψίζονται στον Πίνακα 4.7, παρουσιάζουν ένα εξαιρετικά ενδιαφέρον και αντι-διαισθητικό εύρημα: η προσπάθεια βελτιστοποίησης με τον υβριδικό μηχανισμό αποδείχθηκε, για τα συγκεκριμένα δύσκολα προβλήματα, λιγότερο αποδοτική από τις απλούστερες προσεγγίσεις. Η σύγκριση μεταξύ της ”Hybrid” και της ”Hash only” προσέγγισης στο δύσκολο πρόβλημα

$p08$, όπως οπτικοποιείται στο Σχήμα 4.6, αποδεικνύει το αντι-διαισθητικό εύρημα της ανάλυσής μας.

Πίνακας 4.11: Σύγκριση Απόδοσης των Μηχανισμών Εντοπισμού Διπλότυπων.

Πρόβλημα	**Ρύθμιση**	**Χρόνος (s)**	**Κόμβοι Επεκτ.**
p04	Hybrid	0.029	357
	Hash only	0.026	357
	Bloom only	0.030	357
p08	Hybrid	51.348	106,780
	Hash only	17.100	106,780
	Bloom only	16.717	106,780
p06	Hybrid	>600s	171,011
	Hash only	>600s	818,772
	Bloom only	>600s	1,078,484

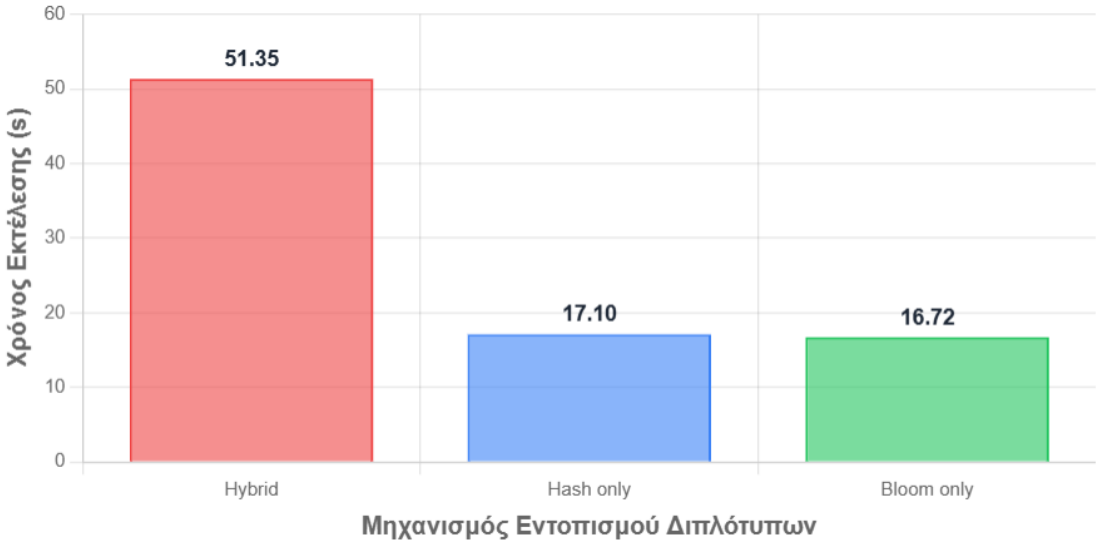
Ερμηνεία των Αποτελεσμάτων

Η ανάλυση των αποτελεσμάτων στον Πίνακα 4.11 αποκαλύπτει ένα εξαιρετικά ενδιαφέρον και αντι-διαισθητικό εύρημα: η θεωρητικά πιο προηγμένη υβριδική μέθοδος αποδεικνύεται στην πράξη η λιγότερο αποδοτική για τα πιο απαιτητικά προβλήματα. Το φαινόμενο αυτό δεν πρέπει να ερμηνευθεί ως μια απλή αποτυχία βελτιστοποίησης, αλλά ως μια εμπειρική απόδειξη του υπολογιστικού κόστους (overhead) που μπορούν να εισάγουν οι σύνθετες δομές δεδομένων, ειδικά σε προβλήματα με πυκνούς χώρους αναζήτησης.

Η σημαντικά πιο αργή απόδοση της υβριδικής μεθόδου στο πρόβλημα p08 (51.35s έναντι 17.10s του "Hash only") μπορεί να αποδοθεί στην υψηλή συχνότητα των "ψευδώς θετικών" (false positives) από το Bloom Filter. Σε έναν πυκνό χώρο όπου πολλές καταστάσεις είναι παρόμοιες, το Bloom filter πιθανότατα απαντά "ίσως ναι" για ένα μεγάλο ποσοστό των νέων κόμβων. Αυτό αναγκάζει τον υβριδικό μηχανισμό να εκτελεί σχεδόν πάντα και τους δύο ελέγχους: τον ταχύτατο, πιθανοτικό έλεγχο στο Bloom filter και τον πιο αργό, ντετερμινιστικό έλεγχο στο Hash Set. Το άθροισμα του χρόνου αυτών των δύο διαδοχικών ελέγχων καταλήγει να είναι μεγαλύτερο από τον χρόνο ενός και μόνο, απευθείας ελέγχου στο Hash Set.

Συμπερασματικά, το πείραμα αυτό αναδεικνύει ένα κρίσιμο trade-off μεταξύ της πολυπλοκότητας μιας δομής και της πρακτικής της απόδοσης. Ενώ ο υβριδικός μηχανισμός σχεδιάστηκε για να μειώσει τις προσπελάσεις στην πιο "ακριβή"

δομή (Hash Set), το ίδιο το κόστος του διπλού ελέγχου υπερκαλύπτει το όφελος στα πιο δύσκολα προβλήματα του 'Rover'. Τα δεδομένα υποδεικνύουν ότι η απλούστερη προσέγγιση της αποκλειστικής χρήσης ενός αποδοτικού Hash Set ("Hash only") αποτελεί την πιο ισορροπημένη και, τελικά, την ταχύτερη λύση για τη βέλτιστη σχεδίαση στο συγκεκριμένο πεδίο, συνδυάζοντας την εγγύηση ακρίβειας με την υψηλή ταχύτητα.



Το γράφημα δείχνει ότι η υβριδική μέθοδος είναι σημαντικά πιο αργή από τις μεθόδους που βασίζονται αποκλειστικά σε Hash Set ή Bloom Filter για το συγκεκριμένο απαιτητικό πρόβλημα.

Σχήμα 4.6: Σύγκριση Χρόνου Εκτέλεσης των Μηχανισμών Εντοπισμού Διπλότυπων στο Πρόβλημα p08

Κεφάλαιο 5

Συμπεράσματα και προτάσεις για μελλοντική βελτίωση

5.1 Ανασκόπηση Σκοπού και Βασικών Ευρημάτων

Η παρούσα πτυχιακή εργασία ξεκίνησε με ένα σαφές ερευνητικό ερώτημα: την ποσοτική διερεύνηση και αξιολόγηση του συμβιβασμού (trade-off) απόδοσης μεταξύ ενός εξειδικευμένου, domain-dependent planner και των σύγχρονων, state-of-the-art domain-independent planners γενικής χρήσης. Για να απαντηθεί αυτό το ερώτημα, σχεδιάστηκε και υλοποιήθηκε ένας πλήρως λειτουργικός, εξειδικευμένος planner για το αριθμητικό πεδίο προβλημάτων "Rover" του IPC 2023. Μέσω μιας ολοκληρωμένης πειραματικής διαδικασίας, όπως αναλύθηκε στο Κεφάλαιο 4, προέκυψε μια σειρά από κρίσιμα συμπεράσματα που θέτουν το πλαίσιο για την τελική, συγκριτική ανάλυση.

Η εσωτερική αξιολόγηση του Κεφαλαίου 4 επιβεβαίωσε την αναποτελεσματικότητα της "τυφλής" αναζήτησης (H0) και ανέδειξε δύο διαμορφώσεις-κλειδιά: τον ταχύτατο satisficing planner **H1+Best-First Search (κάλυψη 9/20)** και τον ισχυρότερο optimal planner H4+A* (κάλυψη 7/20). Αυτά τα ευρήματα, που αποδεικνύουν την αξία τόσο της "άπληστης" όσο και της πληροφοριακής παραδεκτής ευρετικής, θέτουν τη βάση για την κρίσιμη σύγκριση με τα κορυφαία συστήματα γενικής χρήσης.

Αυτά τα εσωτερικά ευρήματα αποδεικνύουν ότι ο εξειδικευμένος planner που αναπτύχθηκε διαθέτει δύο ισχυρές διαμορφώσεις, μία βελτιστοποιημένη για ταχύτητα και μία για εγγυημένη ποιότητα. Με αυτή τη γνώση ως βάση, μπορούμε πλέον να προχωρήσουμε στο επόμενο, και πιο κρίσιμο, βήμα: τη σύγκριση της απόδοσης αυτών των διαμορφώσεων με τους κορυφαίους planners γενικής χρήσης σε παγκόσμιο επίπεδο.

5.2 Συγκριτική Ανάλυση με State-of-the-Art Planners (IPC 2023)

Αφού αξιολογήθηκε η απόδοση των διαφόρων διαμορφώσεων του εξειδικευμένου planner και αναδείχθηκαν οι ισχυρότερες προσεγγίσεις για ικανοποιητική και βέλτιστη σχεδίαση, το επόμενο και κρισιμότερο βήμα είναι η τοποθέτηση αυτών των ευρημάτων στο ευρύτερο πλαίσιο της τρέχουσας ακαδημαϊκής έρευνας. Η παρούσα ενότητα προχωρά σε μια άμεση, ποσοτική σύγκριση της απόδοσης του domain-dependent planner μας με τους κορυφαίους domain-independent planners που διακρίθηκαν στο Numeric Track του Διεθνούς Διαγωνισμού Σχεδιασμού (IPC) του 2023. Σκοπός αυτής της ανάλυσης είναι να απαντηθεί εμπειρικά το κεντρικό ερευνητικό ερώτημα της εργασίας: πόσο μεγάλο, και προς ποια κατεύθυνση, είναι τελικά το χάσμα απόδοσης μεταξύ μιας εξειδικευμένης, "στο χέρι" λύσης και των πιο σύγχρονων συστημάτων γενικής χρήσης. Οι υποενότητες που ακολουθούν θα ορίσουν το πλαίσιο της σύγκρισης, θα παρουσιάσουν τους βασικούς "αντιπάλους" από τον διαγωνισμό και, τέλος, θα παραθέσουν τα συγκριτικά αποτελέσματα.

5.2.1 Πλαίσιο και Μεθοδολογία Σύγκρισης

Για να διασφαλιστεί μια δίκαιη, αντικειμενική και επιστημονικά έγκυρη σύγκριση, η μεθοδολογία που ακολουθείται βασίζεται σε τρεις θεμελιώδεις αρχές.

Πρώτον, τα δεδομένα απόδοσης για τους state-of-the-art, domain-independent planners αντλούνται απευθείας από την επίσημη, δημόσια παρουσίαση των αποτελεσμάτων του Numeric Track του IPC 2023. Αυτό εγγυάται ότι η ανάλυσή μας βασίζεται στα ίδια δεδομένα που χρησιμοποιεί η ακαδημαϊκή κοινότητα για την αξιολόγηση της προόδου στον τομέα.

Δεύτερον, η σύγκριση πραγματοποιείται στο **ακριβές σύνολο των 20 προβλημάτων-αναφοράς (benchmarks)** του πεδίου 'Rover' που χρησιμοποιήθηκαν στον διαγωνισμό, το οποίο αποδείχθηκε ένα από τα πιο απαιτητικά πεδία του, με τους περισσότερους planners να επιτυγχάνουν σχετικά χαμηλές επιδόσεις. Αυτό καθιστά την οποιαδήποτε υπεροχή σε αυτό ακόμα πιο αξιοσημείωτη.

Τρίτον, λόγω της φύσης των συνοπτικών αποτελεσμάτων του IPC, η κύρια μετρική σύγκρισης θα είναι η **κάλυψη (coverage)**, δηλαδή το συνολικό πλήθος των προβλημάτων που κατάφερε να επιλύσει κάθε planner [30]. Αν και αυτό περιορίζει τη δυνατότητα για μια λεπτομερή ανάλυση του χρόνου επίλυσης ή της ποιότητας των παραγόμενων πλάνων, η κάλυψη παραμένει ο πιο ισχυρός και καθιερωμένος δείκτης για τη συνολική ικανότητα και την ευρωστία ενός συστήματος σχεδιασμού. Ωστόσο, είναι κρίσιμο να σημειωθεί ότι το επίσημο σύστημα βαθμολόγησης του IPC δεν βασίζεται αποκλειστικά στην κάλυψη, αλλά συνυπολογίζει και την ποιότητα (κόστος) του παραγόμενου πλάνου, συνήθως μέσω ενός τύπου της μορφής $cost\_min \ / \ cost\_p$. Παρότι δεν έχουμε πρόσβαση

στις αναλυτικές βαθμολογίες, θα ερμηνεύσουμε τα αποτελέσματά μας και υπό το πρίσμα αυτού του ποιοτικού κριτηρίου, εξετάζοντας όχι μόνο αν ο planner μας έλυσε ένα πρόβλημα, αλλά και το πόσο καλά το έλυσε.

Για την παρούσα ανάλυση, θα χρησιμοποιηθούν οι δύο κορυφαίες διαμορφώσεις του planner μας, όπως αυτές αναδείχθηκαν στο Κεφάλαιο 4:

- Για την κατηγορία της **Ικανοποιητικής Σχεδίασης**, θα χρησιμοποιηθεί ο συνδυασμός **H1 + Best-First Search**, ο οποίος επέδειξε την υψηλότερη κάλυψη (9/20).
- Για την κατηγορία της **Βέλτιστης Σχεδίασης**, θα χρησιμοποιηθεί ο συνδυασμός **H4 + A***, ο οποίος αναδείχθηκε ως ο ισχυρότερος βέλτιστος planner μας, επιλύοντας 7 από τα 20 προβλήματα.

Για την πληρότητα της ανάλυσης, υπενθυμίζεται η διάκριση μεταξύ των δύο κατηγοριών. Στην **Ικανοποιητική Σχεδίαση (Satisficing Planning)**, ο πρωταρχικός στόχος είναι η εύρεση μιας οποιασδήποτε έγκυρης λύσης στον συντομότερο δυνατό χρόνο, με την ποιότητα (κόστος) του πλάνου να είναι δευτερεύουσας σημασίας. Αντίθετα, στη **Βέλτιστη Σχεδίαση (Optimal Planning)**, ο στόχος είναι ο εντοπισμός της λύσης με το ελάχιστο δυνατό κόστος, κάτι που απαιτεί όχι μόνο την εύρεση ενός πλάνου, αλλά και την απόδειξη ότι δεν υπάρχει κανένα άλλο με μικρότερο κόστος.

5.2.2 Οι "Αντίπαλοι": Planners Γενικής Χρήσης του IPC 2023

Για να γίνει ουσιαστική η σύγκριση, είναι απαραίτητο να κατανοήσουμε τις θεμελιωδώς διαφορετικές φιλοσοφίες πάνω στις οποίες βασίζονται οι σύγχρονοι domain-independent planners. Οι δύο βασικοί μας "αντίπαλοι" στο πεδίο 'Rover', οι **OMTPlan** και **NLM-CutPlan**, συμμετείχαν στον διαγωνισμό με διακριτές διαμορφώσεις, βελτιστοποιημένες είτε για ταχύτητα (satisficing) είτε για εγγυημένη ποιότητα (optimal).

- Ο **OMTPlan (Parallel)**, νικητής στο 'Rover', ακολουθεί μια **δηλωτική προσέγγιση (declarative approach)**. Μετατρέπει το πρόβλημα PDDL σε ένα μαθηματικό μοντέλο Optimization Modulo Theories (OMT) και αξιοποιεί εξωτερικούς, βελτιστοποιημένους solvers, πιθανότατα σε παραλληλία, για την εύρεση λύσης.
- Ο **NLM-CutPlan (Sat και Orbit)** ακολουθεί την προσέγγιση της ευρετικής αναζήτησης, παρόμοια με τη δική μας. Η κρίσιμη διαφορά είναι ότι οι ευρετικές του παράγονται **πλήρως αυτόματα** από την ανάλυση του PDDL μέσω τεχνικών όπως ο γραμμικός προγραμματισμός, με διακριτές διαμορφώσεις για ικανοποιητική (Sat) και βέλτιστη (Orbit) σχεδίαση.

Η κατανόηση αυτών των στρατηγικών είναι καθοριστική. Ο εξειδικευμένος μας planner θα συγκριθεί με συγκεκριμένες, βελτιστοποιημένες διαμορφώσεις που εκπροσωπούν τις κορυφαίες τεχνολογίες τόσο στη δηλωτική σχεδίαση (OMTPlan) όσο και στην αυτόματη παραγωγή ευρετικών (NLM-CutPlan).

5.2.3 Σύγκριση στην Ικανοποιητική Σχεδίαση (Satisficing)

Στο πεδίο της ικανοποιητικής σχεδίασης, όπου η ταχύτητα εύρεσης μιας οποιασδήποτε έγκυρης λύσης αποτελεί το πρωταρχικό κριτήριο, η ισχυρότερη διαμόρφωση του planner μας είναι ο συνδυασμός **H1 + Best-First Search**. Αυτή η "άπληστη" προσέγγιση αντιπαραβάλλεται με τις αντίστοιχες satisficing διαμορφώσεις των planners του IPC.

Η ανάλυση των επίσημων αποτελεσμάτων του IPC 2023 αποκαλύπτει μια ενδιαφέρουσα εικόνα. Ο παρακάτω πίνακας συνοψίζει την απόδοση σε όρους κάλυψης (coverage):

Πίνακας 5.1: Συγκριτική απόδοση κάλυψης στην Satisficing Σχεδίαση για το πεδίο 'Rover'

Planner	**Προσέγγιση**	**Κάλυψη (Επιλυμένα Προβλήματα από 20)**
OMTPlan (Parallel)	Δηλωτική (Declarative)	**16**
Εξειδικευμένος Planner (H1+GBFS)	**Χειροποίητη Ευρετική**	**9**
NLM-CutPlan Sat	Αυτόματη Ευρετική	3

Τα δεδομένα καταδεικνύουν δύο σαφείς τάσεις. Αφενός, ο **OMTPlan (Parallel)** επιδεικνύει συντριπτική υπεροχή, επιλύοντας 16 από τα 20 προβλήματα. Η declarative προσέγγισή του, ενισχυμένη από την παραλληλία, αποδεικνύεται η πιο αποτελεσματική στρατηγική για ευρεία κάλυψη στο συγκεκριμένο domain.

Αφετέρου, ο εξειδικευμένος μας planner **(9/20)** υπερτερεί σημαντικά του **NLM-CutPlan Sat (3/20)**. Αυτό το εύρημα είναι ιδιαίτερα σημαντικό, καθώς και οι δύο planners βασίζονται στην ευρετική αναζήτηση. Υποδηλώνει ότι η "στο χέρι" ευρετική μας (H1), παρότι απλοϊκή, παρέχει πολύ καλύτερη καθοδήγηση από την αυτόματα παραγόμενη ευρετική του NLM-CutPlan Sat για το συγκεκριμένο πεδίο. Παρόλα αυτά, η συνολική κάλυψη του planner μας παραμένει σημαντικά χαμηλότερη από αυτή του OMTPlan, ένα χάσμα που θα ερμηνευθεί στην ενότητα 5.3.

Είναι, ωστόσο, αξιοσημείωτο ότι, παρότι ο planner μας υπολείπεται σε κάλυψη έναντι του OMTPlan, η ποιότητα των λύσεων που παράγει είναι εξαιρετικά υψηλή, καθώς στις περισσότερες περιπτώσεις εντοπίζει λύσεις με βέλτιστο κόστος. Δεδομένου ότι το σύστημα βαθμολόγησης του IPC συνυπολογίζει την ποιότητα του πλάνου, η υψηλή ποιότητα αυτών των 9 λύσεων θα απέδιδε στον

planner μας υψηλή βαθμολογία, μειώνοντας το συνολικό χάσμα απόδοσης που φαίνεται από την απλή σύγκριση της κάλυψης.

5.2.4 Σύγκριση στη Βέλτιστη Σχεδίαση (Optimal)

Η σύγκριση στη βέλτιστη σχεδίαση αποτελεί την καρδιά της παρούσας εργασίας, καθώς εδώ δοκιμάζεται η ικανότητα ενός planner όχι απλώς να βρει μια λύση, αλλά να εγγυηθεί ότι αυτή είναι η καλύτερη δυνατή. Σε αυτή την κατηγορία, ο ισχυρότερος συνδυασμός μας, **H4 + A***, αντιμετωπίζει τις αντίστοιχες optimal διαμορφώσεις των state-of-the-art planners.

Τα αποτελέσματα, όπως φαίνονται στον Πίνακα 5.2, αντιστρέφουν πλήρως την εικόνα της ικανοποιητικής σχεδίασης και αναδεικνύουν την υπεροχή της εξειδικευμένης προσέγγισης.

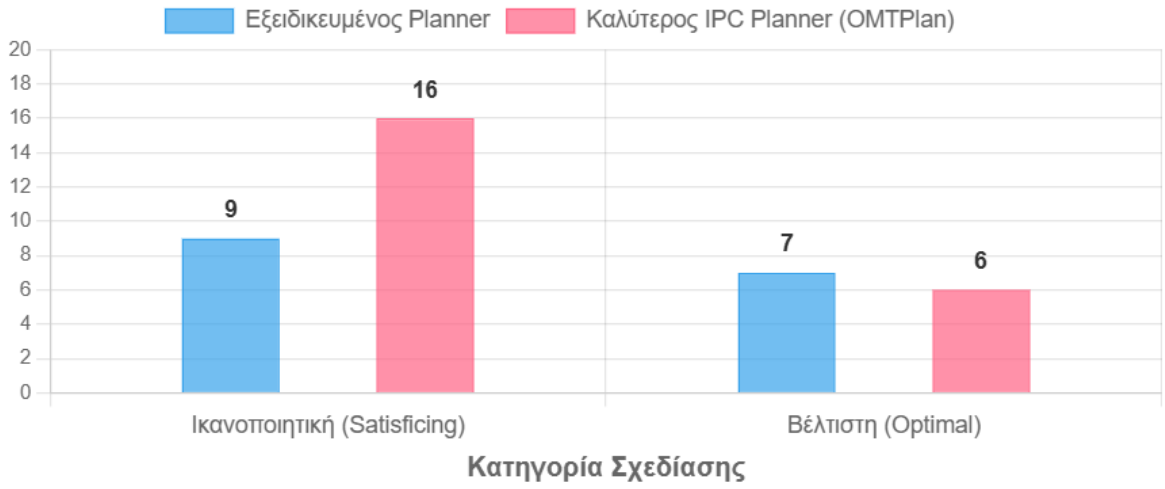
Πίνακας 5.2: Συγκριτική απόδοση κάλυψης στη Optimal Σχεδίαση για το πεδίο 'Rover'

Planner	**Προσέγγιση**	**Κάλυψη (Επιλυμένα Προβλήματα από 20)**
Εξειδικευμένος Planner (H4+A*)	**Χειροποίητη Ευρετική**	**7**
OMTPlan (Parallel)	Δηλωτική (Declarative)	6
NLM-CutPlan Orbit	Αυτόματη Ευρετική	4

Εδώ, ο εξειδικευμένος μας planner όχι μόνο είναι ανταγωνιστικός, αλλά **υπερτερεί**, επιλύοντας 7 από τα 20 προβλήματα. Ξεπερνά τον νικητή του IPC, **OMTPlan (6/20)**, και έχει σημαντικά καλύτερη απόδοση από τον **NLM-CutPlan Orbit (4/20)**.

Το εύρημα αυτό είναι το πιο σημαντικό της πτυχιακής, καθώς παρέχει μια ισχυρή εμπειρική απόδειξη υπέρ της κεντρικής μας υπόθεσης: όταν η απαίτηση για εγγυημένη βελτιστότητα καθιστά την ποιότητα της ευρετικής καθοδήγησης ύψιστης σημασίας, μια καλά σχεδιασμένη, domain-dependent ευρετική μπορεί να αποδειχθεί πιο αποτελεσματική ακόμη και από τις πιο προηγμένες, γενικού σκοπού τεχνικές. Η εξειδικευμένη γνώση που ενσωματώνεται στην ευρετική H4 φαίνεται να παρέχει στον αλγόριθμο A* την ακριβή καθοδήγηση που χρειάζεται για να περιηγηθεί στον τεράστιο χώρο αναζήτησης πιο αποδοτικά από τους αντιπάλους του.

Για την πληρέστερη κατανόηση των ευρημάτων της συγκριτικής ανάλυσης, το Σχήμα 5.1 οπτικοποιεί τη συνολική απόδοση του εξειδικευμένου μας planner έναντι του κορυφαίου planner του IPC, OMTPlan, και στις δύο κατηγορίες σχεδίασης. Το γράφημα αποτυπώνει με σαφήνεια τη διττή φύση του χάσματος απόδοσης: την υστέρηση του εξειδικευμένου planner στην ικανοποιητική σχεδίαση και την καθοριστική του υπεροχή στη βέλτιστη.



Σχήμα 5.1: Συγκριτική Κάλυψη: Εξειδικευμένος Planner vs. Καλύτερος Planner του IPC (OMTPlan)

5.3 Ερμηνεία Αποτελεσμάτων και Απάντηση στο Ερευνητικό Ερώτημα

Η συγκριτική ανάλυση των προηγούμενων ενοτήτων αποκαλύπτει μια σύνθετη και πολυεπίπεδη εικόνα για τη σχέση μεταξύ εξειδικευμένων και γενικών planners. Τα αποτελέσματα, αν και εκ πρώτης όψεως αντιφατικά, όταν ερμηνευθούν υπό το πρίσμα των διαφορετικών φιλοσοφιών σχεδίασης, παρέχουν μια σαφή απάντηση στο κεντρικό ερευνητικό ερώτημα της παρούσας εργασίας.

1. **Η Υστέρηση στην Ικανοποιητική Σχεδίαση: Η Ισχύς της Στρατηγικής Αναζήτησης**

   Στην ικανοποιητική σχεδίαση, τα δεδομένα καταδεικνύουν μια σαφή υπεροχή του OMTPlan (Parallel) όσον αφορά την κάλυψη (coverage), επιλύοντας 16 από τα 20 προβλήματα. Η δηλωτική (declarative) προσέγγισή του, ενισχυμένη από την παραλληλία, αποδεικνύεται η πιο αποτελεσματική στρατηγική για την ευρεία εξερεύνηση του χώρου καταστάσεων στην ικανοποιητική σχεδίαση.

   Ωστόσο, το πιο σημαντικό εύρημα για την παρούσα εργασία είναι η σύγκριση με τον NLM-CutPlan Sat. Ο εξειδικευμένος μας planner (9/20) υπερτερεί συντριπτικά του NLM-CutPlan Sat (3/20). Δεδομένου ότι και οι δύο planners βασίζονται στην ευρετική αναζήτηση, το αποτέλεσμα αυτό υποδηλώνει ότι η ”στο χέρι” ευρετική μας (H1), παρότι απλή στη σύλληψή της, παρέχει πολύ πιο αποτελεσματική καθοδήγηση από την αυτόματα παραγόμενη ευρετική του NLM-CutPlan Sat για το συγκεκριμένο πεδίο.

   Είναι, επιπλέον, κρίσιμο να ερμηνεύσουμε τα αποτελέσματα και υπό το

πρίσμα του επίσημου συστήματος βαθμολόγησης του IPC, το οποίο δεν βασίζεται αποκλειστικά στην κάλυψη. Η ποιότητα (κόστος) του παραγόμενου πλάνου αποτελεί βασικό συστατικό της τελικής βαθμολογίας. Όπως έδειξε η ανάλυση στο Κεφάλαιο 4, ο planner μας, ακόμα και στην "άπληστη" διαμόρφωσή του, εντοπίζει λύσεις εξαιρετικά υψηλής ποιότητας, στην πλειονότητά τους βέλτιστες (κόστος 0 ή 1). Συνεπώς, παρότι η κάλυψη είναι μικρότερη από αυτή του OMTPlan, η εξαιρετικά υψηλή ποιότητα των 9 λύσεων που βρέθηκαν θα απέφερε στον planner υψηλή βαθμολογία στο επίσημο σύστημα του IPC, μειώνοντας έτσι το φαινομενικό χάσμα απόδοσης που προκύπτει από την απλή σύγκριση της κάλυψης.

2. **Η Υπεροχή στη Βέλτιστη Σχεδίαση: Ο Θρίαμβος της Εξειδικευμένης Γνώσης**

   Η νίκη του H4+A* (7/20) έναντι των OMTPlan (6/20) και NLM-CutPlan Orbit (4/20) αποδεικνύει εμπειρικά την τεράστια αξία της εξειδικευμένης γνώσης όταν ο στόχος είναι η εγγυημένη βελτιστότητα.

   Στη βέλτιστη σχεδίαση, ο παράγοντας που καθορίζει την επιτυχία είναι η ικανότητα του αλγορίθμου να "κλαδεύει" όσο το δυνατόν μεγαλύτερα τμήματα του τεράστιου χώρου αναζήτησης. Αυτό επιτυγχάνεται μόνο μέσω μιας εξαιρετικά πληροφοριακής και ταυτόχρονα παραδεκτής ευρετικής. Η ευρετική μας **H4 (Optimal Assignment)** ενσωματώνει ("μεταγλωττίζει") μέσα στη λογική της τη βαθιά, ανθρώπινη κατανόηση της πιο κρίσιμης δομικής ιδιότητας του 'Rover': της δυνατότητας παράλληλης εκτέλεσης εργασιών από τα οχήματα. Πιο συγκεκριμένα, η ικανότητά της να συνυπολογίζει παράλληλες διαδρομές και το ενεργειακό τους κόστος, αντιμετωπίζει κατά μέτωπο τον κεντρικό αριθμητικό συμβιβασμό του 'Rover' που αναλύθηκε στο Κεφάλαιο 3: την ανάγκη για ελαχιστοποίηση των ενεργοβόρων μετακινήσεων και των επαναφορτίσεων, κάτι που οι γενικές ευρετικές φαίνεται πως δυσκολεύονται να μοντελοποιήσουν με την ίδια ακρίβεια.

   Οι γενικοί planners, από την άλλη πλευρά, πρέπει να "ανακαλύψουν" αυτή την πληροφορία δυναμικά. Είτε μέσω της πολύπλοκης μαθηματικής μοντελοποίησης (OMTPlan) είτε μέσω της αυτόματης ανάλυσης του PDDL (NLM-CutPlan), η διαδικασία αυτή είναι υπολογιστικά πολύ πιο "ακριβή". Αυτό το επιπλέον υπολογιστικό κόστος τους οδηγεί σε πρόωρο τερματισμό (timeout) στα πιο δύσκολα προβλήματα, εκεί όπου η στοχευμένη καθοδήγηση του εξειδικευμένου μας planner του επιτρέπει να βρει τη λύση.

3. **Απάντηση στο Ερευνητικό Ερώτημα**

   Τελικά, το "κενό απόδοσης" μεταξύ των δύο προσεγγίσεων είναι υπαρκτό, δυναμικό και η κατεύθυνσή του εξαρτάται από τον στόχο της σχεδίασης.

   - Για την **ικανοποιητική σχεδίαση**, όπου η ευρεία κάλυψη είναι ο στόχος, οι σύγχρονοι planners γενικής χρήσης με προηγμένες στρατηγικές

αναζήτησης (όπως portfolios και declarative solvers) φαίνεται να υπερτερούν.

- Για τη **βέλτιστη σχεδίαση**, ωστόσο, το κενό αντιστρέφεται. Η ποιότητα της ευρετικής καθοδήγησης γίνεται ο κυρίαρχος παράγοντας, και η ικανότητα ενσωμάτωσης βαθιάς, εξειδικευμένης γνώσης σε μια "στο χέρι" ευρετική μπορεί να προσφέρει καθοριστικό πλεονέκτημα απόδοσης, οδηγώντας σε υπεροχή ακόμη και έναντι των πιο σύγχρονων, state-of-the-art συστημάτων.

5.4 Περιορισμοί της Εργασίας

Η ολοκλήρωση κάθε ερευνητικής προσπάθειας συνοδεύεται από την αναγνώριση των ορίων και των περιορισμών της. Η παρούσα εργασία, παρότι πέτυχε τον κεντρικό της στόχο, δεν αποτελεί εξαίρεση. Η παράθεση των παρακάτω περιορισμών είναι απαραίτητη για μια αντικειμενική αποτίμηση της συνεισφοράς της και για τον εντοπισμό μελλοντικών ερευνητικών κατευθύνσεων.

- **Εστίαση σε ένα και μόνο Πεδίο Προβλημάτων**: Ολόκληρος ο σχεδιασμός του planner, και κυρίως των ευρετικών συναρτήσεων, έγινε αποκλειστικά για το πεδίο 'Rover'. Κατά συνέπεια, παρόλο που τα συμπεράσματα για την υπεροχή της εξειδικευμένης προσέγγισης στη βέλτιστη σχεδίαση είναι ισχυρά για το συγκεκριμένο πεδίο, δεν μπορούν να γενικευθούν αυτόματα σε όλα τα αριθμητικά πεδία του Automated Planning. Είναι πιθανό η συγκεκριμένη δομή του 'Rover' να ευνοεί ιδιαίτερα την "στο χέρι" προσέγγιση.

- **Μονολιθική Υλοποίηση**: Ο planner αναπτύχθηκε σε γλώσσα C με στόχο τη μέγιστη απόδοση, αλλά ως ένα μονολιθικό πρόγραμμα και όχι ως ένα ευέλικτο, αρθρωτό framework. Αυτό καθιστά την οποιαδήποτε προσπάθεια προσαρμογής του σε ένα νέο πεδίο προβλημάτων εξαιρετικά δύσκολη, καθώς θα απαιτούσε σημαντική επανεγγραφή του κώδικα.

- **Περιορισμένη Μετρική Σύγκρισης**: Η συγκριτική ανάλυση με τους state-of-the-art planners του IPC βασίστηκε αναγκαστικά στην κάλυψη (coverage), καθώς ήταν η μόνη διαθέσιμη, αντικειμενική μετρική από τα δημοσιευμένα αποτελέσματα. Μια πληρέστερη σύγκριση θα απαιτούσε πρόσβαση σε αναλυτικά δεδομένα, όπως ο ακριβής χρόνος επίλυσης και το κόστος του πλάνου για κάθε πρόβλημα ξεχωριστά, κάτι που δεν ήταν εφικτό.

Παρά τους παραπάνω περιορισμούς, η εργασία επιτυγχάνει τον πρωταρχικό της σκοπό: να προσφέρει μια σαφή, ποσοτικοποιημένη και σε βάθος μελέτη περίπτωσης (case study) που αναδεικνύει εμπειρικά τη δυναμική του συμβιβασμού μεταξύ γενικότητας και εξειδίκευσης στη σύγχρονη σχεδίαση.

5.5 Προτάσεις για Μελλοντική Βελτίωση

Τα συμπεράσματα της παρούσας εργασίας, και ειδικότερα η επιτυχία της εξειδικευμένης προσέγγισης στη βέλτιστη σχεδίαση, ανοίγουν τον δρόμο για μια σειρά από ενδιαφέρουσες επεκτάσεις και βελτιώσεις. Οι προτάσεις που ακολουθούν είναι δομημένες σε τρεις άξονες: τη βελτίωση των ευρετικών συναρτήσεων, τη βελτιστοποίηση της αρχιτεκτονικής του planner και την επέκταση της μεθοδολογίας σε νέα προβλήματα.

- **Υβριδική Ευρετική**: Ανάπτυξη μιας στρατηγικής που θα αξιοποιεί την ταχεία H1 για την εύρεση μιας αρχικής λύσης, το κόστος της οποίας θα χρησιμοποιείται ως άνω φράγμα (upper bound) για μια πιο στοχευμένη αναζήτηση από την ακριβή H4.

- **Βελτιστοποίηση Planner**: Βάσει των ευρημάτων του Κεφαλαίου 4, προτείνεται η απλοποίηση του μηχανισμού εντοπισμού διπλοτύπων με την αποκλειστική χρήση ενός Hash Set, καθώς και η διερεύνηση τεχνικών περιορισμένης μνήμης (π.χ. IDA*) για την αντιμετώπιση μεγαλύτερων προβλημάτων.

- **Επέκταση Μεθοδολογίας**: Εφαρμογή της "στο χέρι" μεθοδολογίας σχεδιασμού ευρετικών σε ένα νέο, διαφορετικό αριθμητικό πεδίο του IPC (π.χ. 'Sailing'), για τον έλεγχο της γενικότητας της προσέγγισης.

- **Αυτόματη Εξαγωγή Γνώσης**: Ως πιο φιλόδοξη κατεύθυνση, η χρήση τεχνικών Μηχανικής Μάθησης (π.χ. Graph Neural Networks) για την αυτόματη εκμάθηση των κρίσιμων δομικών ιδιοτήτων ενός domain, γεφυρώνοντας το χάσμα μεταξύ χειροποίητου σχεδιασμού και πλήρους αυτοματοποίησης.

5.6 Τελικά συμπεράσματα

Η παρούσα πτυχιακή εργασία ξεκίνησε για να διερευνήσει εμπειρικά ένα από τα πιο θεμελιώδη ερωτήματα στον Αυτοματοποιημένο Σχεδιασμό: τον συμβιβασμό μεταξύ της ευελιξίας των planners γενικής χρήσης και της ισχύος των εξειδικευμένων λύσεων. Μέσα από μια ολοκληρωμένη διαδικασία που συνδύασε την υλοποίηση λογισμικού, τον σχεδιασμό αλγορίθμων και την αυστηρή πειραματική ανάλυση, η εργασία προσφέρει μια σαφή και ποσοτικοποιημένη απάντηση.

Η συνεισφορά της είναι τριπλή. **Πρώτον**, σε επίπεδο **υλοποίησης**, κατασκευάστηκε ένας πλήρως λειτουργικός, βέλτιστος domain-dependent planner σε γλώσσα C, ικανός να επιλύει το απαιτητικό αριθμητικό πρόβλημα 'Rover'. **Δεύτερον**, σε επίπεδο **σχεδιασμού**, αναπτύχθηκε και αναλύθηκε μια πρωτότυπη, εξελικτική σειρά πέντε εξειδικευμένων ευρετικών συναρτήσεων (H0-H4), οι οποίες επέδειξαν πώς η σταδιακή ενσωμάτωση βαθιάς γνώσης για τη δομή

του προβλήματος μπορεί να οδηγήσει σε δραματική βελτίωση της καθοδήγησης της αναζήτησης.

Τρίτον, και σημαντικότερο, σε επίπεδο **ανάλυσης**, η εργασία κατάφερε να ποσοτικοποιήσει το χάσμα απόδοσης μεταξύ των δύο προσεγγίσεων. Το τελικό συμπέρασμα είναι αδιαμφισβήτητο: παρότι οι σύγχρονοι planners γενικής χρήσης, με τις προηγμένες στρατηγικές τους, υπερτερούν στην ικανοποιητική σχεδίαση, η εξειδικευμένη γνώση θριαμβεύει στο πιο απαιτητικό πεδίο της βέλτιστης σχεδίασης. Η εμπειρική απόδειξη ότι ο εξειδικευμένος μας planner (H4+A*) έλυσε περισσότερα προβλήματα βέλτιστα (7/20) από οποιονδήποτε state-of-the-art planner στον παγκόσμιο διαγωνισμό IPC 2023, αποτελεί το επιστέγασμα αυτής της προσπάθειας. Αποδεικνύει, τελικά, ότι σε μια εποχή που κυριαρχείται από την αναζήτηση γενικών λύσεων, η βαθιά κατανόηση και η έξυπνη αξιοποίηση των ιδιαιτεροτήτων ενός προβλήματος παραμένει ένας από τους πιο ισχυρούς δρόμους προς την επίτευξη της πραγματικής τεχνητής νοημοσύνης.

Παράρτημα Α

Κώδικας και Τεχνικές Λεπτομέρειες

Το παρόν παράρτημα περιέχει συμπληρωματικό τεχνικό υλικό που αφορά την υλοποίηση της πτυχιακής εργασίας. Σκοπός του είναι να προσφέρει τις απαραίτητες λεπτομέρειες για την πλήρη κατανόηση και την ενδεχόμενη αναπαραγωγή της πειραματικής διαδικασίας. Το παράρτημα περιλαμβάνει τον πλήρη κώδικα PDDL του πεδίου 'Rover', τον πηγαίο κώδικα των ευρετικών συναρτήσεων που αναπτύχθηκαν, καθώς και σύντομες οδηγίες για τη μεταγλώττιση και εκτέλεση του planner[1].

Α .1 Πλήρης Κώδικας PDDL του Πεδίου 'Rover'

Ακολουθεί ο πλήρης ορισμός του πεδίου 'Rover' σε γλώσσα PDDL, όπως αυτός χρησιμοποιήθηκε για την πειραματική διαδικασία της εργασίας.

```
(define (domain rover)
(:requirements :typing :fluents :numeric-fluents :negative-preconditions)
(:types
  rover lander objective camera mode waypoint store - object
)

(:predicates
  (at ?x - rover ?y - waypoint)
  (at_lander ?x - lander ?y - waypoint)
  (can_traverse ?r - rover ?x - waypoint ?y - waypoint)

  (equipped_for_soil_analysis ?r - rover)
  (equipped_for_rock_analysis ?r - rover)
  (equipped_for_imaging ?r - rover)

  (empty ?s - store)
```

[1] Όπως αναφέρθηκε και πριν, όλος ο πηγαίος κώδικας είναι διαθέσιμος στο https://github.com/nikts27/thesis-rover-planner.git

```
  (full ?s - store)

  (have_rock_analysis ?r - rover ?w - waypoint)
  (have_soil_analysis ?r - rover ?w - waypoint)

  (calibrated ?c - camera ?r - rover)
  (supports ?c - camera ?m - mode)
  (available ?r - rover)
  (have_image ?r - rover ?o - objective ?m - mode)

  (visible ?w - waypoint ?p - waypoint)
  (at_soil_sample ?w - waypoint)
  (at_rock_sample ?w - waypoint)

  (on_board ?r - rover)
  (in_sun ?w - waypoint)

  (communicated_soil_data ?w - waypoint)
  (communicated_rock_data ?w - waypoint)
  (communicated_image_data ?o - objective ?m - mode)
)

(:functions
  (energy ?r - rover)
  (recharges)
)

(:action navigate
  :parameters (?x - rover ?y - waypoint ?z - waypoint)
  :precondition (and (can_traverse ?x ?y ?z) (at ?x ?y) (>= (energy ?x)
   8))
  :effect (and (not (at ?x ?y)) (at ?x ?z) (decrease (energy ?x) 8))
)

(:action recharge
  :parameters (?r - rover ?w - waypoint)
  :precondition (and (at ?r ?w) (in_sun ?w))
  :effect (and (assign (energy ?r) 100) (increase (recharges) 1))
)

(:action sample_soil
  :parameters (?x - rover ?s - store ?p - waypoint)
  :precondition (and (at ?x ?p) (at_soil_sample ?p) (
   equipped_for_soil_analysis ?x) (on_board ?s ?x) (empty ?s) (>= (energy
    ?x) 3))
  :effect (and (not (empty ?s)) (full ?s) (have_soil_analysis ?x ?p) (not
    (at_soil_sample ?p)) (decrease (energy ?x) 3))
)
```

```
61
62 (:action sample_rock
63   :parameters (?x - rover ?s - store ?p - waypoint)
64   :precondition (and (at ?x ?p) (at_rock_sample ?p) (
       equipped_for_rock_analysis ?x) (on_board ?s ?x) (empty ?s) (>= (energy
       ?x) 5))
65   :effect (and (not (empty ?s)) (full ?s) (have_rock_analysis ?x ?p) (not
       (at_rock_sample ?p)) (decrease (energy ?x) 5))
66 )
67
68 (:action drop
69   :parameters (?x - rover ?y - store)
70   :precondition (and (on_board ?y ?x) (full ?y))
71   :effect (and (not (full ?y)) (empty ?y))
72 )
73
74 (:action calibrate
75   :parameters (?r - rover ?c - camera ?t - objective ?w - waypoint)
76   :precondition (and (equipped_for_imaging ?r) (calibrated ?c ?r) (
       supports ?c ?m) (visible_from ?t ?w) (at ?r ?w) (>= (energy ?r) 2))
77   :effect (and (not (calibrated ?c ?r)) (decrease (energy ?r) 2))
78 )
79
80 (:action take_image
81   :parameters (?r - rover ?p - waypoint ?o - objective ?c - camera ?m -
       mode)
82   :precondition (and (calibrated ?c ?r) (on_board ?c ?r) (
       equipped_for_imaging ?r) (supports ?c ?m) (visible ?p ?o) (at ?r ?p)
       (>= (energy ?r) 1))
83   :effect (and (have_image ?r ?o ?m) (not (calibrated ?c ?r)) (decrease (
       energy ?r) 1))
84 )
85
86 (:action communicate_soil_data
87   :parameters (?r - rover ?l - lander ?p - waypoint ?x - waypoint ?y -
       waypoint)
88   :precondition (and (at ?r ?x) (at_lander ?l ?y) (have_soil_analysis ?r
       ?p) (visible ?x ?y) (>= (energy ?r) 4))
89   :effect (and (not (have_soil_analysis ?r ?p)) (communicated_soil_data ?
       p) (decrease (energy ?r) 4))
90 )
91
92 (:action communicate_rock_data
93   :parameters (?r - rover ?l - lander ?p - waypoint ?x - waypoint ?y -
       waypoint)
94   :precondition (and (at ?r ?x) (at_lander ?l ?y) (have_rock_analysis ?r
       ?p) (visible ?x ?y) (>= (energy ?r) 4))
```

```
  :effect (and (not (have_rock_analysis ?r ?p)) (communicated_rock_data ?p) (decrease (energy ?r) 4))
)

(:action communicate_image_data
  :parameters (?r - rover ?l - lander ?o - objective ?m - mode ?x - waypoint ?y - waypoint)
  :precondition (and (at ?r ?x) (at_lander ?l ?y) (have_image ?r ?o ?m) (visible ?x ?y) (>= (energy ?r) 6))
  :effect (and (not (have_image ?r ?o ?m)) (communicated_image_data ?o ?m) (decrease (energy ?r) 6))
)
)
```

A .2 Ενδεικτικός Κώδικας Υλοποίησης: Οι Ευρετικές Συναρτήσεις

Στην παρούσα ενότητα παρατίθεται ο πηγαίος κώδικας από το αρχείο `heuristic.h`, ο οποίος αποτελεί τον πυρήνα της πρωτότυπης συνεισφοράς της εργασίας. Ο κώδικας περιλαμβάνει την υλοποίηση της πιο εξελιγμένης παραδεκτής ευρετικής, **H4 (Optimal Assignment)**, καθώς και τις βασικές βοηθητικές συναρτήσεις για τον υπολογισμό του χαλαρού κόστους των στόχων, οι οποίες αξιοποιούνται και από τις απλούστερες ευρετικές.

```c
// Global distance matrix precomputed with Floyd-Warshall
int dist[MAX_ROVERS][MAX_WAYPOINTS][MAX_WAYPOINTS];

// Struct to hold the relaxed cost of achieving a single goal
typedef struct {
    int cost;
    int rover_id; // The best rover to achieve this goal
} GoalCost;

// Comparison function for qsort to sort goals by cost, descending
int compareGoalCosts(const void *a, const void *b) {
    GoalCost *costA = (GoalCost *)a;
    GoalCost *costB = (GoalCost *)b;
    return (costB->cost - costA->cost);
}

// Calculates the minimum relaxed cost for each individual unfulfilled goal
void calculate_all_goal_costs(State *state, GoalCost costs[], int *count) {
    *count = 0;

```

```
    // --- Soil Goals ---
    for (int wp = 0; wp < num_waypoints; wp++) {
        if (!goal.communicated_soil_data[wp] || state->waypoints[wp].communicated_soil) continue;
        for (int r = 0; r < num_rovers; r++) {
            // ... (Implementation details for soil goals)
        }
    }

    // --- Rock Goals ---
    // ... (Similar implementation for rock goals)

    // --- Image Goals ---
    // ... (Similar implementation for image goals)
}

// Calculates the additional cost of recharging for a given assignment
int calculate_energy_cost_for_assignment(State *state, int assigned_costs[MAX_ROVERS]) {
    // ... (Implementation details for energy cost calculation)
}

// The "Optimal Assignment" heuristic (H4)
int heuristic(State nodeState) {
    if (is_solution(nodeState)) return 0;

    GoalCost all_costs[ (MAX_WAYPOINTS * 2 + MAX_OBJECTIVES * MAX_MODES) * MAX_ROVERS ];
    int goal_count = 0;

    calculate_all_goal_costs(&nodeState, all_costs, &goal_count);

    if (goal_count == 0) return 0;

    qsort(all_costs, goal_count, sizeof(GoalCost), compareGoalCosts);

    int h_tasks = 0;
    bool rover_used[MAX_ROVERS] = {false};
    int assigned_costs[MAX_ROVERS] = {0};

    // Greedily assign the most expensive tasks to available rovers
    for (int i = 0; i < goal_count; i++) {
        int rover_id = all_costs[i].rover_id;
        if (rover_id != -1 && !rover_used[rover_id]) {
            h_tasks += all_costs[i].cost;
            assigned_costs[rover_id] = all_costs[i].cost;
            rover_used[rover_id] = true;
        }
    }
```

```
67
68      // Add the admissible energy cost for the assigned tasks
69      int h_energy = calculate_energy_cost_for_assignment(&nodeState,
        assigned_costs);
70
71      if (h_energy == INT_MAX) return INT_MAX;
72
73      return h_tasks + h_energy;
74  }
```

A .3 Οδηγίες Μεταγλώττισης και Εκτέλεσης

Για την αναπαραγωγή της πειραματικής διαδικασίας, ο πηγαίος κώδικας του planner μπορεί να μεταγλωττιστεί και να εκτελεστεί ακολουθώντας τις παρακάτω οδηγίες. Η μεταγλώττιση προτείνεται να γίνει με τον μεταγλωττιστή GCC.

1. **Μεταγλώττιση**

   Από τον κεντρικό φάκελο του project, εκτελέστε την παρακάτω εντολή στο τερματικό. Η εντολή αυτή θα δημιουργήσει ένα εκτελέσιμο αρχείο με το όνομα $rover\_planner$.

   ```
   gcc -O3 -pg planner.c bloom.c -o rover_planner -lm
   ```

   - 03: Ενεργοποιεί το υψηλότερο επίπεδο βελτιστοποίησης για μέγιστη απόδοση.
   - $-pg$: Επιλογή για τη δημιουργία πληροφοριών προφίλ (profiling).
   - $-lm$: Πραγματοποιεί σύνδεση (link) με τη μαθηματική βιβλιοθήκη της C.

2. **Εκτέλεση**

   Το πρόγραμμα εκτελείται από τη γραμμή εντολών, παρέχοντας ως ορίσματα τον επιθυμητό αλγόριθμο, το αρχείο του προβλήματος και το αρχείο εξόδου για τη λύση.

   ```
   ./rover_planner <algorithm> <problem_file> <solution_file>
   ```

3. **Παράδειγμα Εκτέλεσης**:

   Για την επίλυση του προβλήματος $pfile8.pddl$ με τον αλγόριθμο A* και την αποθήκευση της λύσης στο αρχείο $solution8\_astar.txt$, η εντολή θα ήταν:

   ```
   ./rover_planner astar ../problems/pfile8.pddl ../solutions
     ↪ /solution8_astar.txt
   ```

- Το όρισμα $< algorithm >$ μπορεί να πάρει την τιμή $astar$ για βέλτιστη αναζήτηση ή $best$ για άπληστη ικανοποιητική αναζήτηση.

Βιβλιογραφία

[1]: Russell, S. J., Norvig, P. (2021). Artificial Intelligence: A Modern Approach (4th ed.). Pearson.

[2]: Ghallab, M., Nau, D. S., Traverso, P. (2004). Automated Planning: Theory Practice. Morgan Kaufmann.

[3]: Long, D. Fox, M. (2003). The 3rd International Planning Competition: Results and Analysis. Journal of Artificial Intelligence Research, 20, 1-59.

[4]: Fox, M., Long, D. (2003). PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. Journal of Artificial Intelligence Research, 20, 61-124.

[5]: Taitler, A., et al. (2024). The 2023 International Planning Competition. AI Magazine.

[6]: Wilkins, D. E. (1984). Domain-independent planning: Representation and plan generation. Artificial intelligence, 22(3), 269-301.

[7]: Srivastava, B., Kambhampati, S., Mali, A. D. (1998). A Structured Approach for Synthesizing Planners from Specifications. In Proceedings of the 13th IEEE International Conference on Automated Software Engineering.

[8]: Bonet, B., Geffner, H. (2001). Planning as Heuristic Search. Artificial Intelligence, 129(1-2), 5-33.

[9]: Hart, P. E., Nilsson, N. J., Raphael, B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. IEEE Transactions on Systems Science and Cybernetics, 4(2), 100-107.

[10]: Hoffmann, J., Nebel, B. (2001). The FF Planning System: Fast Plan Generation Through Heuristic Search. Journal of Artificial Intelligence Research, 14, 253-302.

[11]: Richter, S., Westphal, M. (2010). The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks.

[12]: Hoffmann, J. (2003). The Metric-FF Planning System: Translating "Ignoring Delete Lists" to Numeric State Variables. Journal of Artificial Intelligence Research, 20, 291-341.

[13]: Scala, E., Haslum, P., Thiebaux, S. (2016). Heuristics for Numeric Planning via Subgoaling. In Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI-16), 3228-3234.

[14]: Scala, E., Haslum, P., Magazzeni, D., Thiebaux, S. (2017). Landmarks for

Numeric Planning Problems. In Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence (IJCAI-17), 4384-4390.

[15]: Coles, A., et al. (2010). A Hybrid Relaxed Planning Graph-LP Heuristic for Numeric Planning.

[16]: Dornhege, C., et al. (2009). Directed Search for Temporal Planning.

[17]: Wilkins, D. E. (1984). Domain-independent planning: Representation and plan generation. Artificial intelligence, 22(3), 269-301.

[18]: Fern, A., Khardon, R., Tadepalli, P. (2011). Learning in planning. In Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques (pp. 509-536). IGI Global.

[19] Srivastava, B., Kambhampati, S., Mali, A. D. (1998). A Structured Approach for Synthesizing Planners from Specifications. In Proceedings of the 13th IEEE International Conference on Automated Software Engineering.

[20]: Fawcett, C., et al. (2011). A new approach to automated planning system configuration.

[21]: Vallati, M., et al. (2013). A new approach for portfolio-based planning.

[22]: https://ipc2023.github.io/

[23]: https://ipc2023-classical.github.io/

[24]: https://ipc2023-learning.github.io/

[25]: https://ataitler.github.io/IPPC2023/

[26]: https://ipc2023-numeric.github.io/

[27]: https://ipc2023-htn.github.io/

[28]: Roveto, F., et al. (2023). NLM-CutPlan: A Numeric Planner with a Cutting-Plane Heuristic. In IPC 2023 Planner Abstracts.

[29]: https://github.com/ipc2023-numeric/ipc2023-dataset/blob/main/rover/domain.pddl

[30]: https://ipc2023-numeric.github.io/results/presentation.pdf

[31]: Leofante F. OMTPlan: A Tool for Optimal Planning Modulo Theories. Journal on Satisfiability, Boolean Modelling and Computation.