Ατομική Διπλωματική Εργασία

ΕΠΕΚΤΑΣΕΙΣ ΕΥΡΕΤΙΚΩΝ ΜΕΘΟΔΩΝ ΓΙΑ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟ ΔΡΑΣΗΣ ΜΕΣΩ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ ΣΥΝΟΛΟΥ ΑΠΑΝΤΗΣΕΩΝ

Μάριος Ιακώβου

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάιος 2022

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Επεκτάσεις Ευρετικών Μεθόδων για Προγραμματισμό Δράσης μέσω Προγραμματισμού Συνόλου Απαντήσεων

Μάριος Ιακώβου

Επιβλέπων Καθηγητής
Δρ. Γιάννης Δημόπουλος

Η Ατομική Διπλωματική Εργασία υποβλήθηκε προς μερική εκπλήρωση των απαιτήσεων απόκτησης του πτυχίου Πληροφορικής του Τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου

Μάιος 2022

Ευχαριστίες

Θα ήθελα να ευχαριστήσω θερμά τον επιβλέποντα καθηγητή της διπλωματικής μου εργασίας κ. Γιάννη Δημόπουλο για τη βοήθεια και τη καθοδήγηση που μου πρόσφερε στα διάφορα στάδια της παρούσας εργασίας. Οι συμβουλές για βελτίωση αλγορίθμων που χρησιμοποιήθηκαν καθώς και οι υποδείξεις όσον αφορά την πειραματική έρευνα ήταν σημαντικές για την ολοκλήρωση της.

Περίληψη

Η παρούσα διπλωματική εργασία μελετά την επέκταση ευρετικών μεθόδων στην επίλυση προβλημάτων Προγραμματισμού Δράσης μέσω Προγραμματισμού Συνόλου Απαντήσεων.

Τα προβλήματα Προγραμματισμού Δράσης με το οποία θα ασχοληθούμε γράφονται σε γλώσσα PDDL. Στα προβλήματα αυτά πρέπει να γίνει μια σειρά δράσεων που να μεταφέρει ένα ευφυή πράκτορα από μια αρχική κατάσταση σε μια κατάσταση στόχου. Προβλήματα Προγραμματισμού Δράσης μπορούν να μοντελοποιηθούν σε προβλήματα Προγραμματισμού Συνόλου Απαντήσεων έτσι ώστε να μπορούν να επιλυθούν με αντίστοιχες τεχνικές.

Στο σύστημα της εργασίας αυτής χρησιμοποιείται η plasp που επιτρέπει την μοντελοποίηση προβλημάτων Προγραμματισμού Δράσης σε προβλήματα Προγραμματισμού Συνόλου Απαντήσεων. Η plasp μεταφράζει PDDL αρχεία σε αντίστοιχα λογικά προγράμματα Προγραμματισμού Συνόλου Απαντήσεων. Στο σύστημα χρησιμοποιούνται συγκεκριμένα ευρετικά στην αναζήτηση για λύσεις, τα οποία αποσκοπούν σε αποτελεσματική επίλυση των προβλημάτων.

Στην εργασία αυτή ασχοληθήκαμε με επεκτάσεις αυτών των ευρετικών μεθόδων του συστήματος, έτσι ώστε να βελτιωθούν οι αναζητήσεις για λύση και να αντιμετωπιστούν συγκεκριμένα προβλήματα που παρατηρήθηκαν κατά την εκτέλεση του συστήματος. Ένα από τα προβλήματα που παρατηρήθηκε ήταν ο εγκλωβισμός του επιλυτή σε αδιέξοδα με αποτέλεσμα να αποτυγχάνει η επίλυση συγκεκριμένων προβλημάτων. Οι επεκτάσεις που υλοποιήθηκαν στα πλαίσια της εργασίας αυτής συνεισφέρουν αύξηση στον αριθμό των προβλημάτων με αδιέξοδα που μπορούν να επιλυθούν. Οι επιδόσεις των παραμέτρων αξιολογήθηκαν με πειραματικές μελέτες και συνδυασμούς μεταξύ των παραμέτρων για επίλυση προβλημάτων.

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

1.1 Σκοπός Διπλωματικής Εργασίας

Ο Προγραμματισμός Δράσης στην Τεχνητή Νοημοσύνη αφορά τη λήψη αποφάσεων από ευφυείς πράκτορες για την επίτευξη κάποιων στόχων. Σε προβλήματα Προγραμματισμού Δράσης πρέπει να γίνει μια σειρά δράσεων που να μεταφέρει ένα ευφυή πράκτορα από μια αρχική κατάσταση σε μια κατάσταση στόχου. Προβλήματα Προγραμματισμού Δράσης μπορούν να μοντελοποιηθούν σε προβλήματα Προγραμματισμού Συνόλου Απαντήσεων για να μπορούν να επιλυθούν με αντίστοιχες τεχνικές.

Το σύστημα που χρησιμοποιείται στα πλαίσια αυτής της διπλωματικής εργασίας μπορεί να κάνει αυτή τη μοντελοποίηση. Στο σύστημα χρησιμοποιούνται συγκεκριμένα ευρετικά στην αναζήτηση για λύσεις, τα οποία αποσκοπούν σε αποτελεσματική επίλυση των προβλημάτων. Ο τρόπος λειτουργίας του συστήματος με περισσότερες λεπτομέρειες θα εξηγηθεί στα αντίστοιχα κεφάλαια στη συνέχεια του κειμένου.

Σκοπός της διπλωματικής εργασίας είναι να γίνουν επεκτάσεις αυτών των ευρετικών μεθόδων του συστήματος με σκοπό την βελτίωση των αποτελεσμάτων στην αναζήτηση λύσεων. Οι επεκτάσεις που έγιναν στα πλαίσια αυτής της διπλωματικής εργασίας επίσης αποσκοπούν στην αντιμετώπιση συγκεκριμένων προβλημάτων που παρατηρήθηκαν κατά την εκτέλεση του συστήματος.

1.2 Δομή Διπλωματικής Εργασίας

Η εργασία αυτή αρχίζει με εισαγωγή σε διάφορες έννοιες του Προγραμματισμού Δράσης και περιγραφής της γλώσσας PDDL που μπορεί να υποστηρίξει την υλοποίηση συγκεκριμένων αλγορίθμων για προβλήματα Προγραμματισμού Δράσης. Έπειτα γίνεται εισαγωγή στον Προγραμματισμό Συνόλου Απαντήσεων και του συστήματος clingo που χρησιμοποιείται για επίλυση προβλημάτων Προγραμματισμού Συνόλου Απαντήσεων.

Στη συνέχεια γίνεται περιγραφή του συστήματος plasp που επιτρέπει την μοντελοποίηση προβλημάτων Προγραμματισμού Δράσης σε προβλήματα Προγραμματισμού Συνόλου Απαντήσεων, το οποίο χρησιμοποιείται στο σύστημα της εργασίας αυτής. Πέραν του συστήματος plasp, γίνεται περιγραφή των μεθόδων που χρησιμοποιούνται στο σύστημα της εργασίας για επίλυση των μοντελοποιημένων αυτών προβλημάτων.

Αφού μελετηθούν αυτές οι έννοιες και περιγραφές λειτουργίας, θα εστιάσουμε σε συγκεκριμένες μεθόδους που χρησιμοποιούνται από το σύστημα τις οποίες θα επεκτείνουμε στα πλαίσια αυτής της διπλωματικής εργασίας. Οι αλλαγές στο σύστημα θα αξιολογηθούν μέσω πειραματικής μελέτης για να εξάγουμε τα αποτελέσματα των επεκτάσεων.

Στο τέλος της εργασίας θα παρουσιαστούν τα συμπεράσματα που προέκυψαν μέσα από την εκπόνηση της καθώς και κάποιες ιδέες για μελλοντική εργασία ως προέκταση της.

Κεφάλαιο 2

Προγραμματισμός Δράσης και Προγραμματισμός Συνόλου Απαντήσεων

2.1 Προγραμματισμός Δράσης

Ο Προγραμματισμός Δράσης (Automated planning and scheduling ή αλλιώς AI Planning) είναι ένα πεδίο Τεχνητής Νοημοσύνης (Artificial Intelligence ή AI) που εστιάζει στην εξερεύνηση διαφόρων αυτόνομων τεχνικών για την υλοποίηση στρατηγικών και ακολουθιών δράσης για ευφυείς πράκτορες (intelligent agents). Στην

ουσία ο Προγραμματισμός Δράσης ασχολείται με αποφάσεις που λαμβάνονται από έναν έξυπνο πράκτορα για να φτάσει σε μια επιθυμητή κατάσταση. Στα προβλήματα Προγραμματισμού Δράσης (planning problems) απαιτείται να βρεθεί μια σειρά ενεργειών (action sequence) που οδηγούν αυτόν τον πράκτορα από μια αρχική κατάσταση (initial state) σε μια επιθυμητή τελική κατάσταση (goal state) όταν εκτελείται [5,8].

Ο κόσμος στα προβλήματα Προγραμματισμού Δράσης αποτελείται από όλα του τα αντικείμενα και την κατάσταση στην οποία βρίσκονται (ατομικά γεγονότα ως μεταβλητές κατάστασης). Μια ενέργεια σε αυτόν τον κόσμο οδηγεί σε μια αλλαγή στην κατάστασή τους, καθιστώντας ορισμένα γεγονότα αληθή (true) ή ψευδή (false). Έτσι, η εκτέλεση μιας ενέργειας έχει επίδραση στον κόσμο. Για να πραγματοποιηθεί μια ενέργεια, πρέπει να πληρούνται ορισμένες προϋποθέσεις. Αυτή η μορφή έκφρασης μας επιτρέπει να βάζουμε τις ενέργειες σε σειρά, καθώς ορισμένες ενέργειες δεν μπορούν να εκτελεστούν πριν από άλλες [5,8]. Για παράδειγμα, ένα άτομο που θέλει να ταξιδέψει σε μια συγκεκριμένη περιοχή με το αυτοκίνητό του, δεν μπορεί να το κάνει πριν ξεκινήσει το αυτοκίνητό του, κάτι που κατ’ επέκταση δεν μπορεί να συμβεί εάν το άτομο δεν έχει πρώτα μπει στο αυτοκίνητο.

Έτσι, πιο συγκεκριμένα, ο Προγραμματισμός Δράσης με την περιγραφή μιας αρχικής κατάστασης του κόσμου (τα αντικείμενά του και τις καταστάσεις τους), μια περιγραφή πιθανών ενεργειών που μπορούν να εκτελεστούν και μια τελική κατάσταση του κόσμου, στοχεύει να δημιουργήσει ένα σχέδιο (plan), το οποίο είναι μια ακολουθία ενεργειών που όταν εκτελεστούν σε σειρά θα συνεχίσουν να μεταμορφώνουν τον κόσμο μέχρι να επιτευχθεί ο στόχος. Το πρόγραμμα που μπορεί να παράγει μια τέτοια ακολουθία ενεργειών, που είναι η λύση ενός προβλήματος Προγραμματισμού Δράσης, ονομάζεται σχεδιαστής (planner) [8].

Για την αποτελεσματική και αποδοτική αντιμετώπιση προβλημάτων Προγραμματισμού Δράσης, χρειαζόταν ο ορισμός μιας γλώσσας που να μπορεί να περιγράψει τέτοια προβλήματα. Αυτό οφείλεται στο γεγονός ότι οι κλασικοί αλγόριθμοι αναζήτησης, όπως η αναζήτηση κατά πλάτος (breadth-first search) ή η αναζήτηση κατά βάθος (depth-first search) δεν μπορούν να χειριστούν καλά πιο σύνθετα προβλήματα, λόγω

του χώρου αναζήτησης (υποσύνολο του συνόλου που αποτελείται από όλες τις καταστάσεις που μπορούν να προσεγγιστούν από μια αρχική κατάσταση) που αυξάνεται εκθετικά, καθώς αυτοί οι αλγόριθμοι εκτελούν μια τυφλή αναζήτηση, χωρίς τη χρήση ευρετικών (heuristics) ή τα ευρετικά που χρησιμοποιούν είναι γενικά και συνεπώς λιγότερο αποτελεσματικά. Μια τέτοια γλώσσα επιτρέπει τη χρήση αποτελεσματικών αλγορίθμων που χρησιμοποιούν ευρετικά για να αποφασίσουν ποια επόμενη κατάσταση θα επιλεχθεί, αντί να γίνεται τυφλή επιλογή. Για να έχουμε καλύτερες ευρετικές συναρτήσεις, τα προβλήματα μπορούν να αναλυθούν σε μικρότερα υποπροβλήματα που μπορούν να επιλυθούν ανεξάρτητα [8,9].

2.2 PDDL

Μια γλώσσα που μπορεί να υποστηρίξει την υλοποίηση αλγορίθμων που μπορούν να χειριστούν πολύπλοκα προβλήματα Προγραμματισμού Δράσης είναι η PDDL (Planning Domain Definition Language). Η PDDL επιτρέπει στους αλγόριθμους Προγραμματισμού Δράσης να επωφεληθούν από τον προαναφερθέντα ορισμό των προβλημάτων Προγραμματισμού Δράσης (δηλαδή στόχοι, καταστάσεις και ενέργειες). Είναι μια προσπάθεια τυποποίησης των γλωσσών για προβλήματα Προγραμματισμού Δράσης, προκειμένου να συγκριθούν επιδόσεις συστημάτων Προγραμματισμού Δράσης [6].

Για την περιγραφή ενός προβλήματος Προγραμματισμού Δράσης στη PDDL χρησιμοποιούνται ένα αρχείο πεδίου εφαρμογής (domain file) και ένα αρχείο προβλήματος (problem file). Το domain file είναι στην ουσία μια περιγραφή του τι υπάρχει στον κόσμο και ποιες ενέργειες μπορούν να γίνουν, ενώ το problem file είναι ένα στιγμιότυπο αυτού του κόσμου που περιγράφει μια αρχική κατάσταση και μια κατάσταση στόχου [6]. Πριν εξετάσουμε ένα πλήρες παράδειγμα ενός προβλήματος Προγραμματισμού Δράσης κωδικοποιημένο σε PDDL, θα δούμε πώς η PDDL περιγράφει τις καταστάσεις, τους στόχους και τις ενέργειες τέτοιων προβλημάτων χρησιμοποιώντας μια τροποποιημένη σύνταξη μιας απλής έκδοσης της PDDL.

2.2.1 Αναπαράσταση Καταστάσεων

Στη PDDL κάθε κατάσταση αναπαρίσταται ως σύζευξη κατηγορημάτων (predicates). Αυτά τα κατηγορήματα υποδηλώνουν αυτό που ισχύει για την τρέχουσα κατάσταση του κόσμου. Η αντιπροσώπευση των καταστάσεων με αυτόν τον τρόπο επιτρέπει τον χειρισμό τους και με πράξεις συνόλων, οι οποίες μερικές φορές μπορεί να είναι επωφελείς [8].

Ως παράδειγμα, μπορούμε να έχουμε μια σύζευξη με τη μορφή προτασιακής λογικής (propositional logic) για να δηλώσουμε ένα άτομο που είναι φιλικό (friendly) και χαρισματικό (charismatic): Friendly ∧ Charismatic. Μπορούμε επίσης να έχουμε μια σύζευξη με τη μορφή λογικής πρώτης τάξης (first-order logic) για να δηλώσουμε δύο διαφορετικούς ανθρώπους που βρίσκονται σε δύο διαφορετικές θέσεις: At(Person1, Place1) ∧ At(Person2, Place2).

2.2.2 Αναπαράσταση Στόχων

Όπως και οι καταστάσεις, οι στόχοι στο PDDL αντιπροσωπεύονται επίσης ως σύζευξη κατηγορημάτων. Ένας στόχος είναι μια μερικώς καθορισμένη κατάσταση και ικανοποιείται από μια προτασιακή κατάσταση εάν αυτή η κατάσταση περιέχει τουλάχιστον όλα τα λεκτικά (literals) σε αυτόν τον στόχο. Ένας στόχος μπορεί επίσης να περιέχει μεταβλητές, οι οποίες αντιμετωπίζονται ως υπαρξιακά ποσοτικοποιημένες (existentially quantified). Ένα πρόβλημα λύνεται όταν βρεθεί μια ακολουθία ενεργειών που καταλήγει σε μια κατάσταση που συνεπάγεται τον στόχο [8,9].

Για να κατανοήσουμε καλύτερα πώς επιτυγχάνεται ένας στόχος, μπορούμε να δούμε μερικά παραδείγματα. Στην περίπτωση που ο στόχος είναι ο σύνδεσμος Friendly ∧ Charismatic, η κατάσταση Friendly ∧ Charismatic ∧ Outgoing ικανοποιεί τον στόχο, επειδή η κατάσταση συνεπάγεται τον στόχο. Όταν ο στόχος περιέχει μια μεταβλητή, για παράδειγμα ο στόχος είναι At(x, Place1) ∧ Truck(x), τότε η κατάσταση Truck(Truck1) ∧ At(Truck1, Place1) συνεπάγεται τον στόχο. Σε αυτήν την περίπτωση, οποιοδήποτε φορτηγό βρίσκεται στο Place1 ικανοποιεί τον στόχο.

2.2.3 Αναπαράσταση Δράσεων

Μια δράση στη PDDL αναπαρίσταται ως ένα σχήμα δράσης (action schema) που αποτελείται από το όνομα της δράσης, το οποίο είναι μια λίστα με όλες τις μεταβλητές που χρησιμοποιούνται στο σχήμα, τις προϋποθέσεις για να συμβεί αυτή η δράση και τα αποτελέσματα αυτής της δράσης. Μια δράση εφαρμόζεται μόνον όταν οι προϋποθέσεις της πληρούνται σε μια κατάσταση. Έτσι μπορούμε να σκεφτούμε τις προϋποθέσεις όπως τους στόχους [8,9].

Για παράδειγμα, υποθέτουμε ότι ένα άτομο θέλει να μετακομίσει από την πόλη που ζει σε μια κοντινή διαφορετική πόλη. Αυτό σημαίνει ότι οι προϋποθέσεις για να ταξιδέψει σε αυτήν την πόλη είναι ότι το άτομο πρέπει να βρίσκεται πρώτα στην πόλη που ζει και η πόλη που ζει να βρίσκεται κοντά στην πόλη στην οποία θέλει να μετακομίσει. Επιπλέον, το άτομο να είναι άνθρωπος και οι πόλεις να είναι πόλεις αποτελούν επίσης προϋποθέσεις, καθώς η δράση προορίζεται για άτομα που μετακινούνται σε πόλεις και όχι με άλλο τρόπο. Αφού πληρούνται αυτές οι προϋποθέσεις, το άτομο μπορεί να μετακομίσει στην νέα πόλη, πράγμα που σημαίνει ότι το άτομο αυτό δεν είναι πλέον παρόν στην πόλη που ζει, αλλά στην νέα πόλη. Ακολουθεί μια αναπαράσταση αυτής της ενέργειας ως σχήμα δράσης:

Action Move(p, from, to),

PRECOND: At(p, from) ∧ Near(from, to) ∧ Person(p) ∧ Town(from) ∧ Town(to)

EFFECT: ¬At(p, from) ∧ At(p, to)

Η αντικατάσταση τιμών για όλες τις μεταβλητές έχει ως αποτέλεσμα την ακόλουθη συγκεκριμένη δράση (ground action):

Action Move(Person1, Town1, Town2),

PRECOND: At(Person1, Town1) ∧ Near(Town1, Town2) ∧ Person(Person1) ∧ Town(Town1) ∧ Town(Town2)

EFFECT: ¬At(Person1, Town1) ∧ At(Person1, Town2)

2.2.4 Εκφραστικότητα και Επεκτάσεις

Υπάρχουν πολλές εκδόσεις της PDDL, η καθεμία με διαφορετικό επίπεδο εκφραστικότητας. Για περισσότερη εκφραστικότητα απαιτούνται πιο σύνθετοι αλγόριθμοι. Είναι σημαντικό η γλώσσα να είναι αρκετά εκφραστική ώστε να περιγράφει μια μεγάλη ποικιλία προβλημάτων, αλλά αρκετά περιοριστική ώστε να επιτρέπει σε αποτελεσματικούς αλγόριθμους να λειτουργούν πάνω της. Το απλούστερο υποσύνολο της PDDL είναι το STRIPS (Stanford Research Institute Planning System) και μια εξέλιξη του, που είναι πιο εκφραστική, είναι η ADL (Action Description Language) [8,9].

Μπορούν να γίνουν κάποιες διακρίσεις μεταξύ των αναπαραστάσεων του STRIPS και ADL. Για παράδειγμα, το STRIPS επιτρέπει μόνο θετικά literals σε καταστάσεις, ενώ η ADL επιτρέπει τόσο θετικά όσο και αρνητικά literals. Επιπλέον, το STRIPS χρησιμοποιεί υπόθεση κλειστού κόσμου, που σημαίνει ότι τυχόν literals που δεν αναφέρονται θεωρούνται ψευδή, ενώ η ADL χρησιμοποιεί υπόθεση ανοιχτού κόσμου, που σημαίνει ότι τυχόν μη αναφερθέντα literals θεωρούνται άγνωστα. Στο STRIPS, μπορούμε να έχουμε μόνο συγκεκριμένα λεκτικά (ground literals) στους στόχους, ενώ στο ADL μπορούμε να έχουμε ποσοτικοποιημένες μεταβλητές. Η ADL έχει κάποιες περαιτέρω εκφράσεις, όπως το δικαίωμα για τις μεταβλητές να έχουν τύπους (types), τις οποίες το STRIPS δεν υποστηρίζει [9].

Οι περιορισμοί που επιβλήθηκαν από την αναπαράσταση του STRIPS είχαν σκοπό να κάνουν τους αλγόριθμους Προγραμματισμού Δράσης απλούστερους και πιο αποτελεσματικούς, επιτρέποντας ταυτόχρονα την περιγραφή πραγματικών προβλημάτων χωρίς δυσκολία. Ωστόσο, αυτοί οι περιορισμοί έχουν δείξει ότι το STRIPS δεν είναι επαρκώς εκφραστικό για ορισμένα πραγματικά πεδία εφαρμογής (domains), γεγονός που οδήγησε στην ανάπτυξη γλωσσικών παραλλαγών όπως η ADL [9].

2.2.5 Παράδειγμα Προβλήματος Προγραμματισμού Δράσης

Για να συνδυάσουμε τις προαναφερθείσες περιγραφές αναπαράστασης στη PDDL, θα κωδικοποιήσουμε ένα πρόβλημα Προγραμματισμού Δράσης από την αρχή μέχρι το τέλος χρησιμοποιώντας τη PDDL.

Ένα συνηθισμένο παράδειγμα ενός προβλήματος Προγραμματισμού Δράσης είναι το "Blocks World". Αυτό το πρόβλημα αποτελείται από κύβους με διαφορετικές ετικέτες που τοποθετούνται σε ένα τραπέζι. Αυτοί οι κύβοι μπορούν επίσης να στοιβάζονται, αλλά μόνο ένας κύβος μπορεί να τοποθετηθεί πάνω από έναν άλλο. Για να σηκώσουμε έναν κύβο σε αυτό το πρόβλημα, ο κύβος δεν πρέπει να έχει άλλον κύβο πάνω του. Θεωρούμε επίσης ότι οι κύβοι έχουν όλοι το ίδιο μέγεθος και χρώμα, ώστε να μην εισάγουμε περισσότερες προϋποθέσεις στο πρόβλημα.

Όπως αναφέρθηκε προηγουμένως, χρησιμοποιούμε ένα domain file και ένα problem file για να περιγράψουμε προβλήματα στη PDDL. Μπορούμε να ξεκινήσουμε με το domain file για να κωδικοποιήσουμε τον κόσμο του προβλήματος που περιγράφεται. Αυτή τη φορά θα χρησιμοποιήσουμε με την κατάλληλη σύνταξη τους ορισμούς που είχαν περιγράφει χρησιμοποιώντας την τροποποιημένη σύνταξη του PDDL. Ακολουθεί το domain file:

```
(define (domain blocksworld)
(:requirements :strips)
(:predicates
        (on ?x ?y) (clear ?x))
(:action move-to-block
        :parameters (?b ?s ?d)
        :precondition (and (clear ?b) (on ?b ?s) (clear ?d))
        :effect
        (and (not (on ?b ?s))
        (not (clear ?d))
        (clear ?s)
        (on ?b ?d)))
```

```
(:action move-to-table
        :parameters (?b ?s)
        :precondition (and (clear ?b) (on ?b ?s))
        :effect
        (and (not (on ?b ?s))
        (clear ?s)
        (on ?b table))))
```

Στην αρχή του domain file, ορίζουμε το όνομα του domain, σε αυτήν την περίπτωση "blocksworld". Το σύνολο απαιτήσεων ενός domain επιτρέπει στον προγραμματιστή να δει εάν μπορεί να χειριστεί τον τομέα. Σε αυτήν την περίπτωση, καθώς δεν δηλώνονται πρόσθετες απαιτήσεις, θα αρκέσει ένας σχεδιαστής αναπαράστασης STRIPS (προεπιλογή).

Ως κατηγορήματα χρησιμοποιούμε (on ?x ?y) για να ορίσουμε ότι το αντικείμενο x βρίσκεται πάνω από το αντικείμενο y. Επιπλέον το κατηγόρημα (clear ?x) ορίζει ότι το αντικείμενο x δεν έχει άλλο αντικείμενο πάνω του.

Δύο ενέργειες είναι δυνατές σε αυτό το πρόβλημα. Η πρώτη ενέργεια είναι η μετακίνηση ενός κύβου από μια αρχική τοποθεσία, που μπορεί να είναι είτε το τραπέζι είτε ένας άλλος κύβος, στην κορυφή ενός κύβου προορισμού: move-to-block(?b ?s ?d). Η δεύτερη ενέργεια είναι η μετακίνηση ενός κύβου από μια αρχική τοποθεσία στο τραπέζι: move-to-table(?b ?s). Για να εκτελεστούν αυτές οι ενέργειες όπως φαίνεται χρειάζονται οι ανάλογες προϋποθέσεις και η εκτέλεση τους επιφέρει τα ανάλογα αποτελέσματα.

Τώρα που έχουμε κωδικοποιήσει το domain file, μπορούμε να προχωρήσουμε στο problem file για να περιγράψουμε ένα στιγμιότυπο του κόσμου. Το problem file πρέπει να περιγράφει την αρχική κατάσταση και την επιθυμητή τελική κατάσταση. Θα χρησιμοποιήσουμε την ακόλουθη αρχική κατάσταση και τελική κατάσταση για το αρχείο προβλήματος όπως φαίνεται στο Σχήμα 2.1.

(α) Αρχική κατάσταση (β) Τελική κατάσταση

Σχήμα 2.1 Αρχική και επιθυμητή τελική κατάσταση για το πρόβλημα

Ακολουθεί το problem file που κωδικοποιεί τις προαναφερθείσες καταστάσεις:

```
(define (problem blocksworld-problem)
        (:domain blocksworld)
        (:objects a b c d table)
        (:init
                (on c b) (on a d) (on b table) (on d table) (clear c) (clear a))
        (:goal
                (and (on a b) (on b table) (on c table) (on d table))))
```

Στην αρχή του problem file, ορίζουμε το όνομα του προβλήματος, σε αυτήν την περίπτωση “blocksworld-problem”. Για αυτό το πρόβλημα θα χρησιμοποιήσουμε το domain "blocksworld" που περιγράψαμε στο domain file. Τα αντικείμενα σε αυτήν την περίπτωση αυτού του κόσμου είναι οι κύβοι a, b, c, d και το τραπέζι table.

Στο μπλοκ :init περιγράφουμε την αρχική κατάσταση του κόσμου σύμφωνα με το Σχήμα 2.1: το c είναι πάνω από το b, το a είναι πάνω από το d, το b και το d βρίσκονται πάνω στο τραπέζι και τα c και a δεν έχουν κάτι επάνω τους. Όταν περιγράφουμε την αρχική κατάσταση, πρέπει να περιγράψουμε όλα τα κατηγορήματα που είναι αληθή.

Στο μπλοκ :goal περιγράφουμε την τελική κατάσταση του κόσμου σύμφωνα με την Εικόνα 1: το a βρίσκεται στην κορυφή του b, τα b, c και d βρίσκονται στο τραπέζι και τα a, c και d δεν έχουν κάτι επάνω τους. Όταν περιγράφουμε την τελική κατάσταση, χρειάζεται μόνο να περιγράψουμε τη διάταξη των κύβων που θέλουμε, λόγω του τρόπου με τον οποίο αναπαρίστανται οι καταστάσεις στη PDDL, όπως αναφέρθηκε προηγουμένως.

Με το domain file και problem file κωδικοποιημένα, μπορούμε να χρησιμοποιήσουμε ένα σχεδιαστή για να μας δώσει μια ακολουθία ενεργειών που μας μεταφέρει από την αρχική κατάσταση στην τελική κατάσταση. Χρησιμοποιώντας τον σχεδιαστή LAMA παίρνουμε την ακόλουθη ακολουθία που είναι και η βέλτιστη λύση:

(move-to-table c b)
(move-to-block a d b)

Δηλαδή είχαμε την απάντηση ότι πρώτα ο κύβος c που βρίσκεται πάνω στον κύβο b πρέπει να μετακινηθεί πάνω στο τραπέζι. Αφού γίνει αυτή η δράση μπορεί να γίνει η επόμενη που είναι να μετακινηθεί ο κύβος a που βρίσκεται πάνω στον κύβο d και να πάει πάνω στον κύβο d. Αυτές οι δράσεις όταν εκτελεστούν θα έχουν τα ανάλογα αποτελέσματα στο κόσμο κάθε φορά όπως προαναφέρθηκε.

2.2.6 Αναζήτηση στον Χώρο Καταστάσεων

Για να λύσουν ένα πρόβλημα, οι σχεδιαστές πραγματοποιούν μια αναζήτηση χώρου κατάστασης. Οι δύο κύριοι τύποι αναζητήσεων είναι η αναζήτηση χώρου καταστάσεων προς τα εμπρός (forward state-space search) και η αναζήτηση σχετικών καταστάσεων προς τα πίσω (backward relevant-states search).

Η αναζήτηση προς τα εμπρός ξεκινά από την αρχική κατάσταση και εφαρμόζει ενέργειες από αυτές που μπορούν να εκτελεστούν με βάση τις προϋποθέσεις, δημιουργώντας νέες καταστάσεις. Αυτή τη διαδικασία δημιουργίας και εξερεύνησης νέων καταστάσεων συνεχίζει μέχρι να βρεθεί μια κατάσταση που συνεπάγεται την κατάσταση στόχου. Ο χώρος αναζήτησης αυτής της αναζήτησης είναι μεγάλος και επομένως θεωρείται ανεπαρκής για επίλυση σύνθετων προβλημάτων [8].

Η αναζήτηση προς τα πίσω ξεκινά από την τελική κατάσταση και εφαρμόζει τις ενέργειες προς τα πίσω μέχρι να βρεθεί μια ακολουθία βημάτων που φτάνει στην αρχική κατάσταση. Οι ενέργειες που επιλέχθηκαν είναι αυτές που σχετίζονται με την τελική κατάσταση. Αυτό είναι ένα σύνολο από όλες τις ενέργειες που έχουν ως αποτέλεσμα τουλάχιστον έναν από τους στόχους, ενώ επίσης δεν αφαιρούν κανέναν

στόχο. Αφού επιλεγεί μια ενέργεια, ο στόχος που βρέθηκε αφαιρείται και όλες οι προϋποθέσεις της ενέργειας προστίθενται στους στόχους. Η διαδικασία επαναλαμβάνεται έως ότου οι στόχοι περιγράφουν την αρχική κατάσταση. Καθώς αυτή η αναζήτηση εφαρμόζει μόνο σχετικές ενέργειες, έχει μικρότερο χώρο αναζήτησης από την αναζήτηση προς τα εμπρός. Επιπλέον, η διαδικασία εύρεσης σχετικών καταστάσεων με την κατάσταση στόχου κατά τη διάρκεια αυτής της αναζήτησης υποβοηθείται από την αναπαράσταση της PDDL [8].

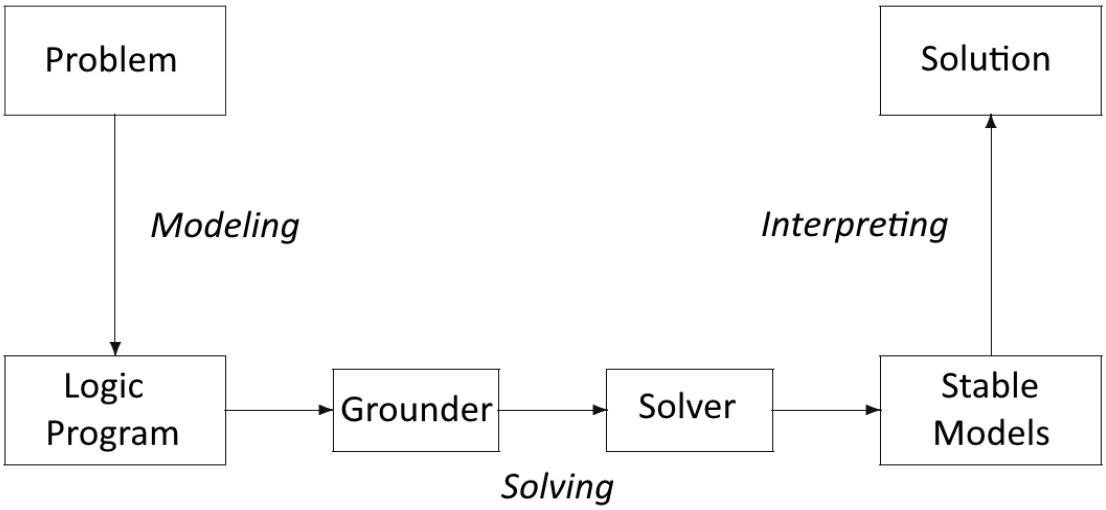
2.3 Προγραμματισμός Συνόλου Απαντήσεων

Ο Προγραμματισμός Συνόλου Απαντήσεων (Answer Set Programming ή ASP) είναι μια προσέγγιση στη δηλωτική επίλυση προβλημάτων, που συνδυάζει μια πλούσια αλλά απλή γλώσσα μοντελοποίησης με ικανότητες επίλυσης υψηλής απόδοσης. Μπορεί να αντιμετωπίζει προβλήματα αναζήτησης NP-hard, όπως ο αυτοματοποιημένος σχεδιασμός, η συστημική βιολογία, η ρομποτική και ο έλεγχος μοντέλων. Ο Προγραμματισμός Συνόλου Απαντήσεων (ΠΣΑ) βασίζεται στη σημασιολογία του σταθερού μοντέλου (stable model) του λογικού προγραμματισμού [2,10].

Στον ΠΣΑ, αντί να πούμε σε έναν υπολογιστή πώς να λύσει ένα πρόβλημα, απλώς περιγράφουμε ποιο είναι το πρόβλημα σε αυτόν. Στη συνέχεια αφήνουμε τον υπολογιστή να βρει τη λύση με βάση την περιγραφή του προβλήματος. Με άλλα λόγια, παρέχουμε μια αναπαράσταση του προβλήματος, αλλά η λύση δίνεται από ένα μοντέλο της αναπαράστασης. Τα προβλήματα εκφράζονται ως λογικά προγράμματα και τα μοντέλα που προκύπτουν είναι τα σύνολα απαντήσεων (answer sets) [10].

Ο ΠΣΑ έχει κάποιες ομοιότητες με τις παραδοσιακές γλώσσες προγραμματισμού όπως η Prolog, η οποία παρέχει μια αναπαράσταση ενός προβλήματος και η λύση δίνεται με την παραγωγή ενός ερωτήματος. Ωστόσο, ενώ στην Prolog ο χρήστης έχει τον πλήρη έλεγχο της εκτέλεσης του προγράμματος, στον ΠΣΑ ο ορισμός ενός προβλήματος και ο τρόπος που θα βρεθεί η λύση είναι αποκομμένοι μεταξύ τους. Ο υπολογισμός στον ΠΣΑ βασίζεται στον αλγόριθμο DPLL, ο οποίος χρησιμοποιείται για την επίλυση του προβλήματος CNF-SAT [2,10].

Για να λυθεί ένα πρόβλημα χρησιμοποιώντας ΠΣΑ, πρέπει πρώτα να μοντελοποιηθεί το πρόβλημα. Χρησιμοποιώντας λογική πρώτης τάξης, το πρόβλημα μπορεί να εκφραστεί ως λογικό πρόγραμμα. Καθώς οι επιλυτές συνόλων απαντήσεων (answer-set solvers) λειτουργούν σε προγράμματα χωρίς μεταβλητές, ένας αντικαταστατής μεταβλητών (grounder) θα πάρει το λογικό πρόγραμμα με μεταβλητές και θα παραγάγει ένα ισοδύναμο πρόβλημα που δεν έχει μεταβλητές. Στη συνέχεια, ο επιλυτής μπορεί να πάρει το πρόγραμμα που προκύπτει και να υπολογίσει τα σύνολα απαντήσεών του. Η λύση στο πρόβλημα προέρχεται από τα ευσταθή μοντέλα (stable models) που κατασκευάστηκαν [2,10]. Αυτή η διαδικασία επίλυσης είναι όπως φαίνεται στο Σχήμα 2.2.



Σχήμα 2.2 Διαδικασία επίλυσης σε ΠΣΑ

Η μεθοδολογία μοντελοποίησης ΠΣΑ βασίζεται σε μια προσέγγιση παραγωγής και δοκιμής (generate and test). Πρώτα παράγει πιθανές υποψήφιες λύσεις και στη συνέχεια ελέγχει εάν αυτές οι υποψήφιες λύσεις παρέχουν μια έγκυρη λύση στο πρόβλημα [10].

2.3.1 Ορισμός Λογικού Προγράμματος και Ευσταθούς Μοντέλου

Ένα λογικό πρόγραμμα P για ένα σύνολο ατόμων (atoms) A είναι ένα πεπερασμένο σύνολο κανόνων (rules).

Ένας κανονικός κανόνας r είναι της μορφής $a_0 \leftarrow a_1, \ldots, a_m, \sim a_{m+1}, \ldots, \sim a_n$ όπου $0 \leq m \leq n$ και κάθε $a_i \in A$ είναι ένα άτομο για $0 \leq i \leq n$. Το a_0 είναι η κεφαλή (head) του κανόνα. Τα $\{a_1, \ldots, a_m, \sim a_{m+1}, \ldots, \sim a_n\}$ είναι το σώμα (body) του κανόνα. $\{a_1, \ldots, a_m\}$ είναι το θετικό σώμα του κανόνα. Τα $\{\sim a_{m+1}, \ldots, \sim a_n\}$ είναι το αρνητικό σώμα του κανόνα.

Αυτή η σύνταξη δεν χρησιμοποιείται από τον πηγαίο κώδικα ΠΣΑ, αλλά ακολουθεί τους ίδιους ορισμούς [10].

Από το λογικό πρόγραμμα θα κατασκευαστούν τα ευσταθή μοντέλα. Ένα σύνολο X από άτομα είναι ένα ευσταθές μοντέλο ενός προγράμματος P εάν το X είναι κλασσικό μοντέλο του P και εάν όλα τα άτομα του X δικαιολογούνται με κάποιο κανόνα στο P (χρησιμοποιώντας υπόθεση κλειστού κόσμου) [10].

2.3.2 Σύνταξη Προγραμματισμού Συνόλου Απαντήσεων

Ο πηγαίος κώδικας για τον ΠΣΑ, παρμένος από το Potassco Teaching [10], χρησιμοποιεί διαφορετικούς συμβολισμούς για τους κανόνες, ωστόσο μπορεί να ακολουθηθεί η συμβολική σύμβαση όπως φαίνεται στο Σχήμα 2.3.

	if	and	or	default negation	classical negation
logic program	:-	,	;	not	-
source code	←	,	;	~	¬

Σχήμα 2.3 Συμβολισμοί για τους κανόνες στον ΠΣΑ

Ένας κανόνας σε σύνταξη ΠΣΑ έχει κεφαλή και σώμα και τελειώνει με τελεία (.). Ένας κανόνας είναι της μορφής head :- $a_1, \ldots, a_n$, not $b_1, \ldots,$ not b_n όπου $0 < n$. Οτιδήποτε πριν το :- είναι η κεφαλή του κανόνα. Τα $\{a_1, \ldots, a_n,$ not $b_1, \ldots,$ not $b_n\}$ είναι το σώμα του κανόνα. Τα $\{a_1, \ldots, a_n\}$ είναι το θετικό σώμα του κανόνα. Τα $\{$not $b_1, \ldots,$ not $b_n\}$ είναι το αρνητικό σώμα του κανόνα.

Εάν το σώμα ενός κανόνα είναι αληθές, τότε η κεφαλή θα είναι επίσης αληθής (στο σύνολο απαντήσεων). Για παράδειγμα, head :- b_1, b_2, b_3. Σε αυτήν την περίπτωση, εάν τα b_1, b_2, και b_3 είναι αληθή, τότε η κεφαλή θα είναι επίσης αληθής.

Ένα σώμα χωρίς κεφαλή είναι ένας περιορισμός ακεραιότητας (integrity constraint), που υποδηλώνει ότι αν το σώμα είναι αληθές, τότε η λύση δεν είναι έγκυρη. Ένα παράδειγμα περιορισμού ακεραιότητας είναι :- b_1, b_2.

Όταν έχουμε κεφαλή χωρίς σώμα όπως στο head., τότε έχουμε ένα γεγονός. Αυτό θεωρείται ως γνώση (knowledge).

Ένας κανόνας επιλογής (choice rule) γράφεται για παράδειγμα ως {h_1; h_2; … ; h_n} :- b_1, … ,b_n. Όταν το σώμα είναι αληθές, τότε θα ισχύει και ένα υποσύνολο των h στην κεφαλή.

Ένας κανόνας πληθικότητας (cardinality rule) γράφεται για παράδειγμα ως 1{h_1; h_2; … ; h_n}3 :- b_1, … ,b_n. Σημαίνει ότι το υποσύνολο των h που μπορεί να είναι αληθές μπορεί να είναι μόνο μεγέθους 1 έως 3. Η πληθικότητα μπορεί επίσης να χρησιμοποιηθεί για το σώμα, για παράδειγμα head :- 1{b_1, … ,b_n}3 σημαίνει ότι για να είναι αληθής η κεφαλή τουλάχιστον 1 b πρέπει να είναι αληθές, αλλά όχι περισσότερα από 3.

Οι μεταβλητές στους κανόνες γράφονται με κεφαλαία γράμματα, για παράδειγμα p(X) :- q(X).

Οι διπλές τελείες (..) υποδεικνύουν μια ακολουθία, για παράδειγμα το p(1..4) είναι το ίδιο με το p(1). p(2). p(3). p(4). Αυτό διευκολύνει τη γραφή μεγάλων συνόλων.

Τα λεκτικά υπό συνθήκη (condition literals) είναι ένας άλλος τρόπος έκφρασης ενός συνόλου. Έχουν τη μορφή l:d όπου l είναι ένα άτομο και d είναι κατηγόρημα πεδίου εφαρμογής (domain predicate). Για παράδειγμα, 1{setColor(v,C):color(C)}1. color(red). color(blue). είναι το ίδιο με το 1{setColor(v,red), setColor(v,blue)}1.

Μπορούν επίσης να χρησιμοποιηθούν συναθροίσματα (aggregates). Για παράδειγμα το συνάθροισμα που χρησιμοποιείται σε έναν κανόνα s(Y) :- r(Y), 2 #sum{X:p(X,Y), q(X)} 7.

Για βελτιστοποίηση μπορούμε να χρησιμοποιήσουμε ελαχιστοποίηση και μεγιστοποίηση. Οι οδηγίες μεγιστοποίησης ή ελαχιστοποίησης δηλώνουν ότι η αναζήτηση πρέπει να γίνεται για να βρεθεί το καλύτερο ευσταθές μοντέλο που δηλώνεται με αυτές. Για παράδειγμα με #maximize{X:area(X)}. το X, το οποίο αναπαριστά ένα εμβαδό, θα μεγιστοποιηθεί στην απάντηση. Άρα, ψάχνουμε το ευσταθές μοντέλο που μπορούμε να βρούμε που θα έχει το μέγιστο δυνατό εμβαδό.

2.4 Clingo

Όπως αναφέρθηκε προηγουμένως, για την επίλυση ενός προβλήματος ΠΣΑ, ένας επιλυτής πρέπει να δημιουργήσει το ευσταθές μοντέλο από την έξοδο ενός αντικαταστατή μεταβλητών, το οποίο είναι μια μορφή του λογικού προγράμματος χωρίς μεταβλητές. Η clingo είναι ένας συνδυασμός αντικατάστασης και επίλυσης που συνδυάζει τις δυνατότητες του gringo αντικαταστατή μεταβλητών και του clasp επιλυτή. Ο συνδυασμός των διαδικασιών αντικατάστασης και επίλυσης μαζί προσφέρει περισσότερο έλεγχο από ό,τι μπορεί να προσφέρει κάθε διαδικασία ξεχωριστά, για παράδειγμα χρησιμοποιώντας σταδιακή αντικατάσταση και επίλυση. Η clingo θα παράγει ένα σύνολο απαντήσεων όταν κληθεί [3,10].

Η clingo είναι εύκολα προσβάσιμη από scripting γλώσσες όπως η Python, η οποία προσφέρει ευκολία στη χρήση για τον χρήστη σε συνδυασμό με τα διαθέσιμα πακέτα.

Για να κατανοήσουμε καλύτερα τις διαδικασίες αντικατάστασης και επίλυσης στον ΠΣΑ, θα εξετάσουμε το gringo αντικαταστατή μεταβλητών και το clasp επιλυτή. ξεχωριστά και στη συνέχεια θα δούμε ένα πλήρες παράδειγμα ενός προβλήματος που θα παρουσιάζει όλα τα στάδια για επίλυση.

2.4.1 Gringo

Το gringo είναι ο αντικαταστατής μεταβλητών που χρησιμοποιείται από τη clingo. Είναι υπεύθυνος να πάρει το λογικό πρόγραμμα που δημιουργεί ο χρήστης με μεταβλητές και να παραγάγει ένα ισοδύναμο ground program χωρίς μεταβλητές. Στην ουσία, ο αντικαταστατής μεταβλητών παράγει μια προτασιακή αναπαράσταση του αρχείου εισόδου. Η έξοδος του δίνεται στη συνέχεια στον επιλυτή για την παραγωγή σταθερών μοντέλων [2,10].

Ένα παράδειγμα διαδικασίας γείωσης είναι δεδομένου του κανόνα p(X) :- q(X). q(1..3). η δημιουργία του προγράμματος p(1) :- q(1). p(2) :- q(2). p(3) :- q(3). Το παραγόμενο πρόγραμμα δεν έχει μεταβλητές (grounded program).

2.4.2 Clasp

Το clasp είναι ο επιλυτής που χρησιμοποιείται από τη clingo για ΠΣΑ. Ο κύριος αλγόριθμός του βασίζεται στην εκμάθηση όρων βασισμένη στις συγκρούσεις (conflict-driven clause learning), η οποία είναι παρόμοια με αυτήν που χρησιμοποιεί ο αλγόριθμος DPLL, που είναι αποτελεσματική στην αντιμετώπιση SAT προβλημάτων. Λόγω αυτού, το clasp μπορεί επίσης να χρησιμοποιηθεί ως διάφοροι επιλυτές, όπως ένας SAT επιλυτής, εκτός από επιλυτής ΠΣΑ. Δεδομένου ενός προγράμματος χωρίς μεταβλητές από το gringo, το clasp θα υπολογίσει τα ευσταθή μοντέλα (σύνολα απαντήσεων) [2,10].

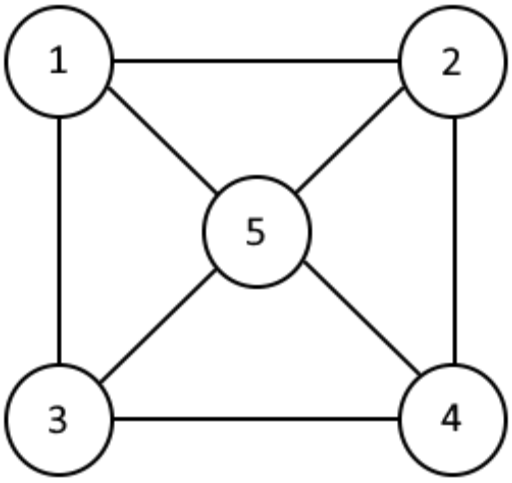
2.4.3 Παράδειγμα Προβλήματος Προγραμματισμού Συνόλου Απαντήσεων

Για να δούμε τον ΠΣΑ στην πράξη, θα δημιουργήσουμε ένα λογικό πρόγραμμα και θα δούμε τα αποτελέσματα που δίνονται από την αντικατάσταση μεταβλητών του gringo και την επίλυση του clasp.

Εμπνευσμένοι από τα παραδείγματα στο Potassco Teaching [10], θα χρησιμοποιήσουμε το πρόβλημα “Vertex Coloring”, το οποίο είναι ένα πρόβλημα χρωματισμού γράφου. Ο στόχος είναι να αντιστοιχιστούν χρώματα στους κόμβους του γραφήματος, έτσι ώστε

να μην υπάρχουν δύο κόμβοι που συνδέονται με μια ακμή που να έχουν το ίδιο χρώμα. Κάθε κόμβος μπορεί να έχει μόνο ένα χρώμα.

Το πρώτο βήμα για τη σύνταξη του λογικού προγράμματος είναι να γίνει διάκριση μεταξύ του στιγμιότυπου του προβλήματος (problem instance) και της κλάσης του προβλήματος (problem class). Σε αυτήν την περίπτωση, το στιγμιότυπο του προβλήματος αποτελείται από τον γράφο, ο οποίος αποτελείται από κόμβους και ακμές, μαζί με τα χρώματα που πρέπει να αντιστοιχίσουμε στους κόμβους. Θα χρησιμοποιήσουμε τρία χρώματα για το πρόβλημά μας: κόκκινο, πράσινο και μπλε, μαζί με ένα μη-κατευθυνόμενο γράφο με πέντε κορυφές και τις ακμές (1,2), (1,3), (1,5), (2, 4), (2,5), (3,4), (3,5), (4,5). Ο γράφος του στιγμιότυπου του προβλήματος που θα λύσουμε φαίνεται στο Σχήμα 2.4.



Σχήμα 2.4 Μη-κατευθυνόμενος γράφος με κόμβους και ακμές

Οι κόμβοι, οι ακμές και τα χρώματα αντιπροσωπεύονται από σύνολα γεγονότων και για αυτά χρησιμοποιούμε τα κατηγορηματικά σύμβολα vertex, edge και color (κατηγορήματα vertex/1, edge/1 και color/1). Το αρχείο προβλήματος μπορεί να γραφτεί ως εξής:

```
vertex(1..5).
edge(1,2). edge(1,3). edge(1,5).
edge(2,4). edge(2,5).
```

```
edge(3,4). edge(3,5).
edge(4,5).
color(red). color(green). color(blue).
```

Αφού ο γράφος είναι μη κατευθυνόμενος, δεν χρειάζεται να χρησιμοποιήσουμε τα δεδομένα edge(1,2) και edge(2,1) μαζί για παράδειγμα, καθώς έχουν την ίδια σημασία εδώ.

Αφού αναπαραστήσουμε το στιγμιότυπο του προβλήματος, μπορούμε να μεταβούμε στη κλάση του προβλήματος. Αυτό το πρόβλημα χρωματισμού γραφήματος μπορεί να εκφραστεί με τους περιορισμούς ότι κάθε κόμβος πρέπει να έχει μόνο ένα χρώμα και ότι δύο συνδεδεμένοι κόμβοι δεν πρέπει να έχουν το ίδιο χρώμα. Κωδικοποιώντας αυτό σε ΠΣΑ, το κωδικοποιημένο αρχείο μπορεί να γραφτεί ως εξής:

```
1 { colored(X,C) : color(C) } 1 :- vertex(X).
:- edge(X,Y), colored(X,C), colored(Y,C).
```

Ο πρώτος κανόνας πληθικότητας εκφράζει την πρώτη δήλωση για τους κόμβους που πρέπει να έχουν ακριβώς ένα χρώμα ο καθένας. Ο περιορισμός ακεραιότητας που ακολουθεί εκφράζει ότι οι συνδεδεμένοι κόμβοι δεν πρέπει να έχουν το ίδιο χρώμα, καθώς όταν το σώμα είναι αληθές, τότε η λύση (στην ουσία δύο συνδεδεμένοι κόμβοι που έχουν το ίδιο χρώμα) δεν είναι έγκυρη. Το κατηγόρημα colored (colored/2) μας δίνει ένα χρωματισμό για ένα κόμβο.

Η πρώτη γραμμή μπορεί να θεωρηθεί ως γεννήτρια, η οποία θα δημιουργήσει πιθανές υποψήφιες λύσεις, ενώ η δεύτερη γραμμή είναι ένας ελεγκτής που φιλτράρει τις έγκυρες λύσεις από τις δημιουργούμενες υποψήφιες λύσεις. Αυτό ισχύει για τη μεθοδολογία μοντελοποίησης του ΠΣΑ όπως αναφέρθηκε προηγουμένως, η οποία βασίζεται σε μια προσέγγιση δημιουργίας και δοκιμής.

Τα αρχεία που κωδικοποιήθηκαν μπορούν τώρα να παραμείνουν ξεχωριστά ή να γίνουν ένα ενωμένο αρχείο. Τρέχοντας τον gringo αντικαταστατή μεταβλητών πάνω στο αρχείο με παράμετρο --text μας δίνει έξοδο που είναι αναγνώσιμη. (Η έξοδος κανονικά

θα άλλαζε γραμμή μετά από κάθε τελεία, αλλά για λόγους σαφήνειας θα εμφανίσουμε συμπιεσμένα τα αποτελέσματα.)

Ο αντικαταστατής μεταβλητών εξάγει τα κατηγορήματα των ακμών, μαζί με τα κατηγορήματα χρώματος:

```
edge(1,2). edge(1,3). edge(1,5).
edge(2,4). edge(2,5).
edge(3,4). edge(3,5).
edge(4,5).
color(red). color(green). color(blue).
```

Εκτός από τα γεγονότα που ήταν δεδομένα, ο αντικαταστατής μεταβλητών μας δίνει μια επέκταση του εύρους 1..5 για το κατηγόρημα της κορυφής ως γεγονότα για κάθε κορυφή:

```
vertex(1). vertex (2). vertex (3). vertex(4). vertex(5).
```

Προερχόμενοι από τον πρώτο περιορισμό πληθικότητας που είχαμε στο αρχείο του λογικού προγράμματος, λαμβάνουμε το εξής, το οποίο μας λέει ότι κάθε κορυφή μπορεί να έχει μόνο ένα χρώμα:

```
1 {colored(1,red), colored(1,blue), colored(1,green)} 1.
1 {colored(2,red), colored(2,blue), colored(2,green)} 1.
1 {colored(3,red), colored(3,blue), colored(3,green)} 1.
1 {colored(4,red), colored(4,blue), colored(4,green)} 1.
1 {colored(5,red), colored(5,blue), colored(5,green)} 1.
```

Από τον περιορισμό ακεραιότητας που είχαμε στο αρχείο του λογικού προγράμματος, ο αντικαταστατής μεταβλητών αφήνει μόνο τα άτομα που έχουν να κάνουν με το χρωματισμό, αφαιρώντας την παρουσία του κατηγορήματος ακμής καθώς γνωρίζουμε ήδη την τιμή του, αφού δόθηκε ως γεγονός:

:- colored(1,red);colored(2,red).
:- colored(1,blue);colored(2,blue).
:- colored(1,green);colored(2,green).
:- colored(1,red);colored(3,red).
…
:- colored(4,blue);colored(5,blue).
:- colored(4,green);colored(5,green).

Ο αντικαταστατής μεταβλητών δεν γνωρίζει εάν ένας κόμβος θα χρωματιστεί με ένα συγκεκριμένο χρώμα, καθώς αυτό πρέπει να καθοριστεί από τον επιλυτή. Έτσι, τα έγχρωμα κατηγορήματα εδώ δεν μπορούν να απλοποιηθούν και πρέπει να περάσουν στον επιλυτή.

Τώρα που είδαμε τι κάνει το grounding σε αυτήν την περίπτωση, μπορούμε να υπολογίσουμε τα ευσταθή μοντέλα που παραγόμενου προγράμματος χωρίς μεταβλητές, περνώντας το αποτέλεσμα στο clasp και χρησιμοποιώντας την επιλογή 0 για να πάρουμε όλες τις απαντήσεις.

Εκτελώντας αυτό θα εκτυπωθούν τα κατηγορήματα edge, color και vertex, μαζί με τα colored κατηγορήματα που είναι η λύση. Για λόγους επίδειξης θα εξετάσουμε μόνο τις λύσεις που βρέθηκαν αντί για τις άλλες πληροφορίες εξόδου. Η έξοδος μόνο του χρωματισμού γίνεται χρησιμοποιώντας τη γραμμή #show colored/2. στο λογικό μας πρόγραμμα εκ των προτέρων, υποδεικνύοντας στον επιλυτή να εμφανίζει μόνο περιπτώσεις του δυαδικού κατηγορήματος colored. Οι απαντήσεις που βρέθηκαν είναι οι εξής:

colored(1,red) colored(2,green) colored(3,green) colored(4,red) colored(5,blue)

colored(1,red) colored(2,blue) colored(3,blue) colored(4,red) colored(5,green)

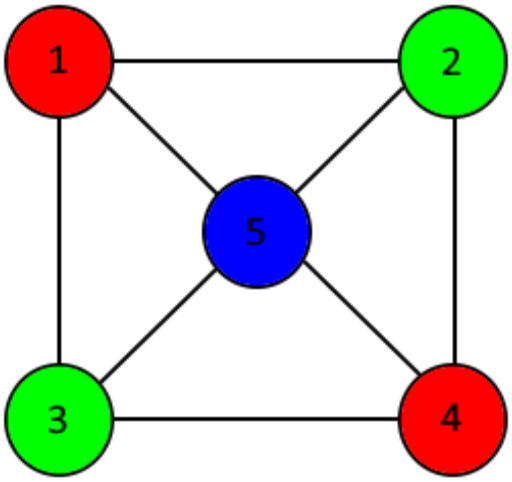
colored(1,green) colored(2,red) colored(3,red) colored(4,green) colored(5,blue)

colored(1,green) colored(2,blue) colored(3,blue) colored(4,green) colored(5,red)

colored(1,blue) colored(2,red) colored(3,red) colored(4,blue) colored(5,green)

colored(1,blue) colored(2,green) colored(3,green) colored(4,blue) colored(5,red)

Ο επιλυτής δίνει επίσης έξοδο "SATISFIABLE", που σημαίνει ότι έχει βρεθεί μια λύση. Όπως φαίνεται από τις απαντήσεις που παράγονται, ο επιλυτής έχει παράγει έγκυρους χρωματισμούς του γράφου ως λύσεις. Οπτικοποιώντας την πρώτη απάντηση παίρνουμε τον γράφο που φαίνεται στο Σχήμα 2.5.



Σχήμα 2.5 Χρωματισμός του γράφου που παράχθηκε από τον επιλυτή

Οι εντολές που έτρεξαν ξέχωρα με gringo και clasp μπορούν να τρέχουν χρησιμοποιώντας μόνο μια εντολή clingo για απλοποίηση [10].

Κεφάλαιο 3

Plasp: Επίλυση Προβλημάτων Προγραμματισμού Δράσης μέσω Προγραμματισμού Συνόλου Απαντήσεων

3.1 Plasp

Το σύστημα που χρησιμοποιείται στα πλαίσια της διπλωματικής εργασίας χρησιμοποιεί ΠΣΑ για επίλυση προβλημάτων Προγραμματισμού Δράσης. Για να επιτευχθεί αυτό χρειάζεται μια μοντελοποίηση προβλημάτων Προγραμματισμού Δράσης σε προβλήματα ΠΣΑ. Η μετάφραση των domain και problem PDDL αρχείων για κάθε ξεχωριστό πρόβλημα σε αντίστοιχα λογικά προγράμματα ΠΣΑ κάθε φορά δεν είναι ιδιαίτερα χρονικά αποτελεσματική. Για αυτό τον λόγο χρειάζεται ένας μεταφραστής που να το κάνει αυτόματα.

Αυτό μπορεί να γίνει με χρήση του μεταφραστή plasp. Η plasp παίρνει τα domain και problem files σε μορφή PDDL και τα μεταφράζει σε μορφή ΠΣΑ. Η μορφή εξόδου της plasp αποτελείται από μεταβλητές κατάστασης που τροποποιούνται από ενέργειες εάν πληρούνται οι προϋποθέσεις τους. Όπως και στη PDDL, ο στόχος είναι να επιτευχθεί

ένας συγκεκριμένος στόχος ξεκινώντας από μια αρχική κατάσταση, εκτελώντας μια ακολουθία ενεργειών [1, 4].

Για να γίνει κατανοητή η λειτουργία της plasp θα γίνει μια επισκόπηση στη σύνταξη και σημασιολογία του αρχείου εξόδου που παράγει, και έπειτα θα αναλυθεί ένα παράδειγμα μετάφρασης ενός αρχείου PDDL με χρήση της plasp.

3.1.1 Σύνταξη και Σημασιολογία Εξόδου Plasp

Για τη σύνταξη και σημασιολογία της εξόδου της plasp θα επικεντρωθούμε σε έννοιες που χρησιμοποιήθηκαν και στο παράδειγμα του κεφαλαίου 2 “Blocks World”, έτσι ώστε να μπορέσουμε στη συνέχεια να δούμε ένα παράδειγμα εξόδου μετάφρασης εκείνου του προβλήματος.

Η plasp για να δηλώσει μια μεταβλητή χρησιμοποιεί variable(variable<name>)). Για να βάλουμε μια τιμή στο πεδίο ορισμού μια μεταβλητής χρησιμοποιούμε contains(<variable>, <value>).

Οι ενέργειες στη plasp δηλώνονται με action(action(<name>)). Για να δοθεί μια προϋπόθεση στην ενέργεια ότι μια μεταβλητή πρέπει να έχει κάποια τιμή πριν εκτελεστεί χρησιμοποιούμε precondition(<action>, <variable>, <value>). Το αποτέλεσμα της ενέργειας δίνει μια τιμή σε μια μεταβλητή. Αν η ενέργεια έχει αποτέλεσμα υπό επιπλέον συνθήκη, τότε δηλώνεται με postcondition(<action>, effect(<number>), <variable>, <value>). Αλλιώς χρησιμοποιείται στη θέση του effect(<number>) το effect(unconditional).

Για να δοθούν οι αρχικές τιμές στα κατηγορήματα που είναι αληθή στην αρχική κατάσταση χρησιμοποιείται το initialState(<variable>, <value>). Η επιθυμητή τιμή ενός κατηγορήματος στην τελική κατάσταση δηλώνεται με goal(<variable>, <value>).

Στη PDDL αν χρησιμοποιήσουμε προδιαγραφές “typing”, τότε μπορούμε να δηλώσουμε τύπους στα αντικείμενα. Για υποστήριξη αυτού στην plasp μπορούμε να δηλώσουμε τύπους με χρήση type(type<name>)). Αν ένας τύπος κληρονομεί έναν άλλο

μπορούμε να το δηλώσουμε με inherits(<type 1>, <type 2>). Για να δηλώσουμε ότι μια σταθερά έχει ένα συγκεκριμένο τύπο χρησιμοποιείται το has(<constant>, <type>). Η δήλωση μιας σταθεράς γίνεται με το constant(constant(<name>)).

Η plasp επίσης υποστηρίζει παράγωγες μεταβλητές (derived variables), παράγωγα κατηγορήματα (derived predicates), mutex groups και κανόνες αξιώματος (axiom rules) [4].

3.1.2 Παράδειγμα Μετάφρασης PDDL Αρχείου με Plasp

Αφού είδαμε βασικά κομμάτια της σύνταξης και σημασιολογίας στη plasp, μπορούμε να δούμε ένα παράδειγμα εξόδου που θα δώσει η plasp μετά από μετάφραση PDDL αρχείου σε λογικό πρόγραμμα ΠΣΑ.

Θα χρησιμοποιήσουμε το πρόβλημα “Blocks World” που χρησιμοποιήθηκε στο υποκεφάλαιο 2.1.5 του κεφαλαίου 2, με τελική κατάσταση στο πρόβλημα να είναι αυτή στο Σχήμα 2.1 που παρουσιάστηκε προηγουμένως.

Για κωδικοποίηση του domain file θα χρησιμοποιήσουμε αυτή τη φορά απαιτήσεις “typing” για να δώσουμε τύπους στα αντικείμενα κύβους και τραπέζι. Το domain file μπορεί να γραφτεί ως εξής:

```
(define (domain blocksworld)
(:requirements :typing)
(:types
        block
        table
)
(:predicates
        (on ?x - block ?y - block) (on ?x - block ?z - table) (clear ?x - block))
(:action move-fb-tb
        :parameters (?b - block ?s - block ?d - block)
        :precondition (and (clear ?b) (on ?b ?s) (clear ?d))
```

```
        :effect
        (and (not (on ?b ?s))
        (not (clear ?d))
        (clear ?s)
        (on ?b ?d)))
(:action move-ft-tb
        :parameters (?b - block ?s - table ?d - block)
        :precondition (and (clear ?b) (on ?b ?s) (clear ?d))
        :effect
        (and (not (on ?b ?s))
        (not (clear ?d))
        (on ?b ?d)))
(:action move-fb-tt
        :parameters (?b - block ?s - block ?d - table)
        :precondition (and (clear ?b) (on ?b ?s))
        :effect
        (and (not (on ?b ?s))
        (clear ?s)
        (on ?b ?d))))
```

Το domain τώρα διαχωρίζει την ενέργεια μετακίνησης ενός κύβου στην κορυφή ενός άλλου κύβου σε μετακίνηση είτε από κύβο πάνω σε κύβο σε άλλο κύβο είτε από κύβο πάνω στο τραπέζι σε κύβο. Αυτό γίνεται αφού τώρα τα αντικείμενα χρησιμοποιούν τύπους είτε κύβου με block είτε τραπεζιού με table.

Η κωδικοποίηση του problem file για το πρόβλημα που προαναφέραμε λαμβάνοντας υπόψη τις αλλαγές στις απαιτήσεις θα γίνει ως εξής:

```
(define (problem blocksworld-problem)
(:domain blocksworld)
(:objects
        a - block
        b - block
```

```
        c - block
        d - block
        t - table
)
(:init
        (on c b) (on a d) (on b t) (on d t) (clear c) (clear a))
(:goal
        (and (on a b) (on b t) (on c t) (on d t))))
```

Αφού κωδικοποιήσαμε τα αρχεία μπορούμε να προχωρήσουμε στην ανάλυση της μετάφρασης που θα κάνει η plasp. Η μορφή εξόδου της plasp είναι μορφής ΠΣΑ της οποίας η σύνταξη εξηγήθηκε σε προηγούμενο κεφάλαιο. Θα χωρίσουμε την έξοδο σε σημεία για την ανάλυση.

Αρχικά μπορούμε να δούμε τη κωδικοποίηση του domain μέσα στο αρχείο. Πρώτα η plasp δηλώνει τις τιμές true και false σαν boolean τιμές που θα χρησιμοποιηθούν:

```
boolean(true).
boolean(false).
```

Έπειτα δηλώνει τους τύπους block και table καθώς και την έννοια της κληρονόμησης τύπου:

```
type(type("block")).
type(type("table")).
has(X, type(T2)) :- has(X, type(T1)), inherits(type(T1), type(T2)).
```

Για κωδικοποίηση των κατηγορημάτων όπως τα δηλώσαμε στο domain file γίνεται χρήση των ακόλουθων:

```
variable(variable(("on", X1, X2))) :- has(X1, type("block")), has(X2, type("block")).
variable(variable(("on", X1, X2))) :- has(X1, type("block")), has(X2, type("table")).
variable(variable(("clear", X1))) :- has(X1, type("block")).
```

Εδώ δηλώθηκαν τα κατηγορήματα ότι ένας κύβος μπορεί να βρίσκεται πάνω σε άλλο κύβο, ένας κύβος μπορεί να βρίσκεται πάνω στο τραπέζι και ότι ένας κύβος μπορεί να μην έχει κάτι επάνω του. Τα κατηγορήματα αυτά θεωρούνται μεταβλητές εδώ αφού μπορούν να πάρουν τιμή true ή false και αυτό δηλώνεται με:

```
contains(X, value(X, B)) :- variable(X), boolean(B).
```

Έπειτα ακολουθεί η κωδικοποίηση της δράσης για μετακίνηση ενός κύβου από πάνω από έναν κύβο στην κορυφή άλλου κύβου:

```
action(action(("move-fb-tb", X1, X2, X3))) :- has(X1, type("block")), has(X2, type("block")), has(X3, type("block")).

precondition(action(("move-fb-tb", X1, X2, X3)), variable(("clear", X1)), value(variable(("clear", X1)), true)) :- action(action(("move-fb-tb", X1, X2, X3))).
precondition(action(("move-fb-tb", X1, X2, X3)), variable(("on", X1, X2)), value(variable(("on", X1, X2)), true)) :- action(action(("move-fb-tb", X1, X2, X3))).
precondition(action(("move-fb-tb", X1, X2, X3)), variable(("clear", X3)), value(variable(("clear", X3)), true)) :- action(action(("move-fb-tb", X1, X2, X3))).

postcondition(action(("move-fb-tb", X1, X2, X3)), effect(unconditional), variable(("on", X1, X2)), value(variable(("on", X1, X2)), false)) :- action(action(("move-fb-tb", X1, X2, X3))).
postcondition(action(("move-fb-tb", X1, X2, X3)), effect(unconditional), variable(("clear", X3)), value(variable(("clear", X3)), false)) :- action(action(("move-fb-tb", X1, X2, X3))).
postcondition(action(("move-fb-tb", X1, X2, X3)), effect(unconditional), variable(("clear", X2)), value(variable(("clear", X2)), true)) :- action(action(("move-fb-tb", X1, X2, X3))).
postcondition(action(("move-fb-tb", X1, X2, X3)), effect(unconditional), variable(("on", X1, X3)), value(variable(("on", X1, X3)), true)) :- action(action(("move-fb-tb", X1, X2, X3))).
```

Όπως φαίνεται για να γίνει η ενέργεια πρέπει να έχουμε έναν κύβο που βρίσκεται πάνω σε αντικείμενο κύβου και θα μετακινηθεί στην κορυφή ενός άλλου αντικειμένου κύβου. Οι προϋποθέσεις της δράσης φαίνονται στη συνέχεια αφού για να εκτελεστεί χρειάζεται να μην έχει καταρχάς κάποιο κύβο πάνω του ο κύβος που θα μετακινηθεί, να βρίσκεται πάνω στον δεύτερο κύβο και ο κύβος στον οποίο θα μετακινηθεί να μην έχει κάτι από πάνω του. Εφόσον εκτελεστεί αυτή η δράση θα έχουμε αποτέλεσμα να γίνει ψευδές ότι ο κύβος βρίσκεται πάνω από τον κύβο στον οποίο βρισκόταν πριν και ότι η τοποθεσία στην οποία μετακινήθηκε ο κύβος είναι πλέον ελεύθερη (ο προηγούμενος κύβος που τώρα βρίσκεται από κάτω). Η τοποθεσία που βρισκόταν πριν ο κύβος τώρα είναι ελεύθερη (ο προηγούμενος κύβος που ήταν κάτω από αυτόν που μετακινήθηκε) και τώρα ο κύβος βρίσκεται πάνω από τον άλλο κύβο στον οποίο μετακινήθηκε.

Ομοίως μεταφράζεται και η δράση να μετακινήσουμε ένα κύβο από το τραπέζι πάνω σε άλλο κύβο:

action(action(("move-ft-tb", X1, X2, X3))) :- has(X1, type("block")), has(X2, type("table")), has(X3, type("block")).

precondition(action(("move-ft-tb", X1, X2, X3)), variable(("clear", X1)), value(variable(("clear", X1)), true)) :- action(action(("move-ft-tb", X1, X2, X3))).
precondition(action(("move-ft-tb", X1, X2, X3)), variable(("on", X1, X2)), value(variable(("on", X1, X2)), true)) :- action(action(("move-ft-tb", X1, X2, X3))).
precondition(action(("move-ft-tb", X1, X2, X3)), variable(("clear", X3)), value(variable(("clear", X3)), true)) :- action(action(("move-ft-tb", X1, X2, X3))).

postcondition(action(("move-ft-tb", X1, X2, X3)), effect(unconditional), variable(("on", X1, X2)), value(variable(("on", X1, X2)), false)) :- action(action(("move-ft-tb", X1, X2, X3))).
postcondition(action(("move-ft-tb", X1, X2, X3)), effect(unconditional), variable(("clear", X3)), value(variable(("clear", X3)), false)) :- action(action(("move-ft-tb", X1, X2, X3))).

postcondition(action(("move-ft-tb", X1, X2, X3)), effect(unconditional), variable(("on", X1, X3)), value(variable(("on", X1, X3)), true)) :- action(action(("move-ft-tb", X1, X2, X3))).

Εδώ για να γίνει η ενέργεια πρέπει να έχουμε έναν κύβο που βρίσκεται πάνω σε αντικείμενο τραπεζιού και θα μετακινηθεί στην κορυφή ενός άλλου αντικειμένου κύβου. Οι προϋποθέσεις είναι οι ίδιες με την προηγούμενη δράση. Τα αποτελέσματα της δράσης εδώ είναι τα ίδια με τη διαφορά ότι μετακινείται από τραπέζι, άρα δεν χρειάζεται να δηλώσουμε το τραπέζι ως ελεύθερο.

Για τη μετακίνηση ενός κύβου από την κορυφή ενός άλλου κύβου πάνω στο τραπέζι η κωδικοποίηση είναι παρόμοια.

Προχωρώντας μπορούμε να δούμε την κωδικοποίηση του προβλήματος μέσα στο αρχείο. Αρχικά δηλώνονται τα αντικείμενα a,b,c,d και t του προβλήματος που αντιπροσωπεύουν τους τέσσερις κύβους και το τραπέζι, άρα θα έχουν τύπους block και table:

```
constant(constant("a")).
has(constant("a"), type("block")).

constant(constant("b")).
has(constant("b"), type("block")).

constant(constant("c")).
has(constant("c"), type("block")).

constant(constant("d")).
has(constant("d"), type("block")).

constant(constant("t")).
has(constant("t"), type("table")).
```

Έπειτα δηλώνεται η αρχική κατάσταση των κατηγορημάτων με υπόθεση κλειστού κόσμου μετά τις δηλώσεις:

```
initialState(variable(("on", constant("c"), constant("b"))), value(variable(("on", constant("c"), constant("b"))), true)).
initialState(variable(("on", constant("a"), constant("d"))), value(variable(("on", constant("a"), constant("d"))), true)).
initialState(variable(("on", constant("b"), constant("t"))), value(variable(("on", constant("b"), constant("t"))), true)).
initialState(variable(("on", constant("d"), constant("t"))), value(variable(("on", constant("d"), constant("t"))), true)).
initialState(variable(("clear", constant("c"))), value(variable(("clear", constant("c"))), true)).
initialState(variable(("clear", constant("a"))), value(variable(("clear", constant("a"))), true)).

initialState(X, value(X, false)) :- variable(X), not initialState(X, value(X, true)).
```

Εδώ παρατηρούμε ότι ξεκινούμε με τον κύβο c πάνω στον b, τον κύβο a πάνω στον d και τους κύβους b και d πάνω στο τραπέζι. Επίσης, οι κύβοι c και a δεν έχουν κάτι από πάνω τους. Έπειτα έχουμε δήλωση για την υπόθεση κλειστού κόσμου όπου θεωρούμε ψευδές οτιδήποτε δεν δηλώθηκε ως αληθές.

Τέλος έχουμε τη δήλωση της τελικής κατάστασης με τη δήλωση των στόχων:

```
goal(variable(("on", constant("a"), constant("b"))), value(variable(("on", constant("a"), constant("b"))), true)).
goal(variable(("on", constant("b"), constant("t"))), value(variable(("on", constant("b"), constant("t"))), true)).
goal(variable(("on", constant("c"), constant("t"))), value(variable(("on", constant("c"), constant("t"))), true)).
goal(variable(("on", constant("d"), constant("t"))), value(variable(("on", constant("d"), constant("t"))), true)).
```

Θέλουμε ο κύβος a να βρίσκεται πάνω στον κύβο b και οι κύβοι b,c και d να βρίσκονται πάνω στο τραπέζι. όπως φαίνεται στις δηλώσεις.

3.1.3 Επίλυση Μεταφρασμένου Προβλήματος

Το μεταφρασμένο πρόβλημα από τη plasp τώρα χρειάζεται να επιλυθεί. Αυτό μπορεί να γίνει με χρήση της clingo και ενός meta encoding. Το meta encoding μας επιτρέπει να τρέξουμε δράσεις και παράλληλα εκτός από σειριακά. Με την ιδέα των παράλληλων δράσεων μπορούμε να εφαρμόσουμε χαλαρώσεις στη διαδικασία της αναζήτησης για λύση. Μπορούμε να δούμε κάποια κατηγορήματα αυτού του meta encoding που χρησιμοποιεί το πρόγραμμα για την αναπαράσταση του κόσμου για περαιτέρω εξερεύνηση.

Με το κατηγόρημα fluent(X) έχουμε την κατάσταση X στην οποία μπορεί να βρίσκεται ένα αντικείμενο του κόσμου ή τη σχέση που μπορεί να συνδέει διάφορα αντικείμενα και η τιμή αληθείας της οποίας μπορεί αν αλλάζει στον χρόνο. Στο παράδειγμα που θέσαμε με το “Blocks World” θα είχαμε για παράδειγμα fluent(variable(“on”, X1,X2)). Κατ’ επέκταση το κατηγόρημα fluent(X,V) περιέχει μια κατάσταση X στην οποία μπορεί να βρίσκεται ένα αντικείμενο του και επιπλέον την τιμή αληθείας V της κατάστασης αυτής. Αυτό μας δείχνει αυτή η κατάσταση ισχύει ή όχι στον κόσμο. Για παράδειγμα με το προηγούμενο fluent αν μπορεί να είναι είτε false είτε true, τότε έχουμε τα κατηγορήματα fluent(variable(“on”, X1,X2), value(variable(“on”, X1,X2), false)) και fluent(variable(“on”, X1,X2), value(variable(“on”, X1,X2), true)).

Με το κατηγόρημα holds(X,V,t) έχουμε μια κατάσταση X στην οποία μπορεί να βρίσκεται ένα αντικείμενο του κόσμου και την τιμή αληθείας V της κατάστασης αυτής στη χρονική στιγμή t. Για παράδειγμα για χρονική στιγμή t θα έχουμε holds(variable(“on”, X1,X2), value(variable(“on”, X1,X2), true), t). Με αυτό τον τρόπο αποθηκεύουμε τη κατάσταση που βρίσκεται ο κόσμος σε κάθε χρονική στιγμή. Το κατηγόρημα holds υποχρεωτικά δίνει μόνο μια τιμή για την τιμή αληθείας, δηλαδή ένα αντικείμενο δεν μπορεί να έχει δύο τιμές αληθείας true και false ταυτόχρονα σε μια χρονική στιγμή.

Για δράσεις έχουμε το κατηγόρημα occurs(A,t) που περιέχει τη δράση A που εκτελείται τη χρονική στιγμή t. Με αυτό τον τρόπο αποθηκεύουμε όλες τις δράσεις που γίνονται σε κάθε χρονική στιγμή. Για παράδειγμα για χρονική στιγμή t θα έχουμε occurs(action(("move-fb-tt", X1, X2, X3)), t).

Αν θέλουμε να δείξουμε ότι ένα αντικείμενο πρέπει να μπορεί να επηρεαστεί μόνο από μια δράση και όχι από πολλές ταυτόχρονα χρησιμοποιούμε το κατηγόρημα single(X, t), όπου έχουμε μια κατάσταση X στην οποία μπορεί να βρίσκεται ένα αντικείμενο του κόσμου στη χρονική στιγμή t. Με αυτό τον τρόπο δεν υπάρχει παράλληλη επεξεργασία ενός αντικειμένου από πολλές δράσεις ταυτόχρονα.

Για καταστάσεις που μπορούν να αλλάξουν τιμή αληθείας κατά την εκτέλεση του προγράμματος χρησιμοποιείται το κατηγόρημα produce(X,V), όπου έχουμε μια κατάσταση (σχέση) X στην οποία μπορεί να βρίσκεται ένα αντικείμενο (ή ένα σύνολο αντικειμένων) του κόσμου και την τιμή αληθείας V της κατάστασης αυτής. Αντίθετα για καταστάσεις των οποίων η τιμή αληθείας παραμένει σταθερή έχουμε το κατηγόρημα persist(X,V).

Οι στόχοι αποθηκεύονται στο κατηγόρημα goal(X,V), στο οποίο έχουμε μια κατάσταση X στην οποία θέλουμε να βρίσκεται ένα αντικείμενο του κόσμου και την τιμή αληθείας V της κατάστασης αυτής.

Εκτός από τα κατηγορήματα μπορούμε να δούμε και κάποιους από τους κανόνες που χρησιμοποιούνται στο πρόγραμμα. Πρώτα μπορούμε να δούμε πως δημιουργούνται όλες οι πιθανές καταστάσεις στον κόσμο. Αυτό γίνεται μέσω των κανόνων:

```
fluent(X,V) :- produce(X,V).
fluent(X,V) :- persist(X,V).
fluent(X,V) :- initialState(X,V), fluent(X).
fluent(X,V) :- active(A), postcondition(A,X,V), fluent(X).
fluent(X) :- fluent(X,V).
```

Ο πρώτος κανόνας δημιουργεί fluents από καταστάσεις που μπορεί να βρεθεί ένα αντικείμενο που μπορούν να αλλάξουν τιμή αληθείας κατά τη διάρκεια του προγράμματος. Ο δεύτερος κανόνας κάνει το ίδιο αλλά με τιμές αληθείας που δεν μπορούν να αλλάξουν. Ο τρίτος κανόνας δημιουργεί fluents από την αρχική κατάσταση του κόσμου. Ο τέταρτος κανόνας δημιουργεί fluents από τα αποτελέσματα δράσεων όταν εκτελεστούν. Ο πέμπτος κανόνας δημιουργεί fluents όπου δεν μας ενδιαφέρει η τιμή αληθείας τα οποία θα χρησιμοποιηθούν για συσχετίσεις καταστάσεων κατηγορημάτων.

Για να δημιουργηθεί η αρχική κατάσταση του κόσμου με τα κατηγορήματα που προαναφέραμε χρησιμοποιούμε το κατηγόρημα holds με t ίσο με 0, δηλαδή holds(X,V,0) :- initialState(X,V), fluent(X).

Για να εξασφαλίσουμε ότι κάθε αντικείμενο έχει μόνο μια κατάσταση στην αρχή και να αποφύγουμε τη δημιουργία αντιφάσεων χρησιμοποιούμε τον κανόνα :- fluent(X), #count{V : holds(X,V,0)} > 1. Για να εξασφαλίσουμε το ίδιο σε άλλες χρονικές στιγμές εκτός της αρχικής χρησιμοποιούμε τον κανόνα 1 {holds(X,V,t) : fluent(X,V)} 1 :- fluent(X).

Ένας κανόνας που χρησιμοποιείται για τις δράσεις είναι ο selfdefeat(A,X) :- active(A), precondition(A,X,V), _parallel = 1, has_condition(A,X,1), not postcondition(A,X,V). Αυτός ο κανόνας δημιουργεί κατηγορήματα selfdefeat(A,X), όπου A μια δράση που μπορεί να εκτελεστεί στον κόσμο και X μια κατάσταση του ενός αντικειμένου του κόσμου όταν με την εκτέλεση μιας δράσης αλλάζει η τιμή αληθείας της κατάστασης του.

Για δημιουργία κατηγορημάτων single(X,t) για κάθε κατάσταση αντικειμένου που αλλάζει μετά από μια δράση σε μια χρονική στιγμή t χρησιμοποιούμε τον κανόνα single(X,t) :- occurs(A,t), selfdefeat(A,X). Για να αποτρέψουμε την παράλληλη εκτέλεση αυτών των δράσεων την ίδια χρονική στιγμή χρησιμοποιούμε τον περιορισμό :- single(X,t), #count{A : occurs(A,t), postcondition(A,X,V), not precondition(A,X,V)} > 1.

Για να αποφύγουμε την αδράνεια στο σύστημα χρησιμοποιούμε τον περιορισμό :- planner_on = 0, not occurs(A,t) : active(A). Αν η σταθερά planner_on έχει την τιμή 0, τότε σε κάθε χρονική στιγμή θα εκτελεστεί κάποια δράση.
Για να ελέγχουμε ότι οι δράσεις που εκτελούνται στον κόσμο όντως έδωσαν τα αποτελέσματα που αναμένονταν χρησιμοποιούμε τον περιορισμό :- occurs(A,t), postcondition(A,X,V), fluent(X), not holds(X,V,t). Δεν μπορεί να εκτελεστεί κάποια δράση χωρίς να υπάρχει το ανάλογο κατηγόρημα holds(X,V,t).

Για να επιβεβαιώσουμε ότι κάθε αλλαγή στον κόσμο έχει προκύψει από τα αποτελέσματα της εκτέλεσης μιας δράσης χρησιμοποιούμε τον περιορισμό :- _inertia = 0, holds(X,V,t), not holds(X,V,t-1), not occurs(A,t) : active(A), postcondition(A,X,V), not precondition(A,X,V). Κάθε κατάσταση στον χρόνο t πρέπει να ήταν αποτέλεσμα μιας δράσης στον χρόνο t-1, δηλαδή να μην υπήρχε ήδη στην προηγούμενη χρονική στιγμή.

Στο πρόγραμμα υπάρχει η σταθερά που ρυθμίζει το επίπεδο παραλληλίας #const _parallel = X, όπου τιμές 1-4 δίνουν επίπεδα παραλληλίας και για άλλες τιμές το πρόγραμμα εκτελείται σειριακά. Για αποτροπή της εκτέλεσης πολλαπλών ενεργειών κατά την σειριακή εκτέλεση έχουμε τον περιορισμό :- _parallel != 0, _parallel != 1, _parallel != 2, _parallel != 3, _parallel != 4, #count{A : occurs(A,t)} > 1.

Για να εξετάσουμε αν ο κόσμος βρίσκεται σε μια κατάσταση στο χρόνο t που αποτελεί μια τελική κατάσταση όπου οι στόχοι του προβλήματος έχουν επιτευχθεί χρησιμοποιούμε τον κανόνα :- query(t), goal(X,V), not holds(X,V,t). Με αυτό ελέγχουμε αν έχουμε αποθηκευμένες με το κατηγόρημα holds(X,V,t) όλες τις καταστάσεις του κατηγορήματος goal(X,V).

Κάποιοι άλλοι σημαντικοί κανόνες για το σύστημα θα εξηγηθούν σε επόμενα υποκεφάλαια γιατί τροποποιούνται για χρήση ευρετικών μεθόδων στην επίλυση.

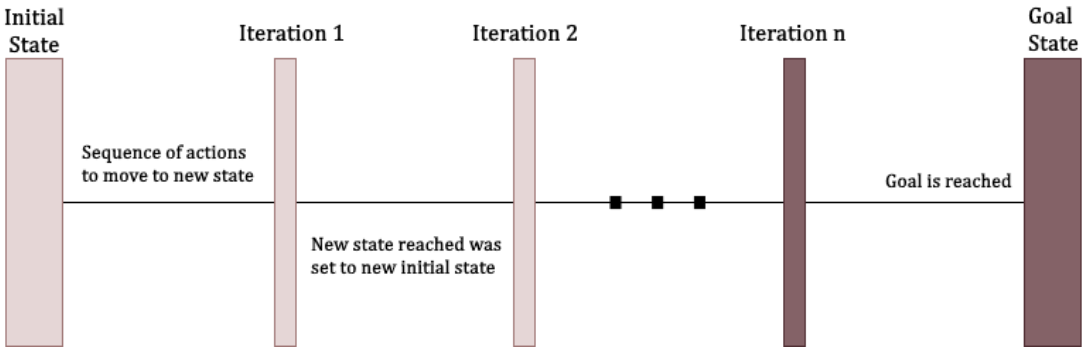
3.2 Χρήση Ευρετικών Μεθόδων στη Plasp

Μια επέκταση του συστήματος που προαναφέραμε είναι με την χρήση ευρετικών μεθόδων στην επίλυση. Η χρήση ευρετικών μπορεί να επιφέρει γρηγορότερη λύση στο πρόβλημα αφού μειώνει τον χώρο αναζήτησης λύσεων. Αυτό είναι αρκετά σημαντικό αφού η επίλυση προβλημάτων Προγραμματισμού Δράσης με χρήση της plasp είναι ιδιαίτερα χρονοβόρα. Μέσω περιορισμών και μεγιστοποίησης ή ελαχιστοποίησης τιμών επηρεάζουμε τον τρόπο που ένας έξυπνος πράκτορας θα εκτελεί κινήσεις στον κόσμο. Τα ευρετικά όμως δεν μας εγγυώνται ότι θα πάρουμε την καλύτερη ή πιο αποδοτική λύση, καθώς η λύση που παίρνουμε με χρήση ευρετικού μπορεί να μην είναι η βέλτιστη.

Μια ιδέα της χρήσης ευρετικών για το σύστημα ήταν να δοκιμάσουμε να φτάσουμε σε ένα ενδιάμεσο σημείο του πλάνου, από το οποίο να μπορούμε έπειτα να προχωρήσουμε στο τελικό πιο εύκολα. Αυτή η ιδέα θα επεκταθεί στα επόμενα υποκεφάλαια για να δούμε πως μπορούν να εφαρμοστούν τέτοιου είδους ευρετικά.

3.2.1 Διαχωρισμός Προβλήματος σε Υποπροβλήματα και Επίλυση

Πριν αναλύσουμε την ιδέα των ευρετικών μεθόδων στο σύστημα, ας δούμε πρώτα την ιδέα του διαχωρισμού του προβλήματος σε υποπροβλήματα. Μεγάλα και δύσκολα προβλήματα μπορούν να μοιραστούν σε μικρότερα και ίσως ευκολότερα προβλήματα. Με κάθε υποπρόβλημα θέλουμε να φτάνουμε πιο κοντά στη λύση του αρχικού προβλήματος. Αυτό μπορεί να γίνει με την μετακίνηση από την κατάσταση (state) που φτάσαμε με επίλυση ενός προηγούμενου υποπροβλήματος, σε μια νέα κατάσταση που είναι πιο κοντά στην λύση του αρχικού προβλήματος, μέσω μιας σειράς από δράσεις. Αυτή η νέα κατάσταση θα είναι η κατάσταση από την οποία θα ξεκινήσουμε για να προχωρήσουμε σε μια άλλη επόμενη κατάσταση σε επόμενη επανάληψη, μέχρι να φτάσουμε στη λύση του αρχικού προβλήματος. Αυτός ο διαχωρισμός φαίνεται στο Σχήμα 3.1.



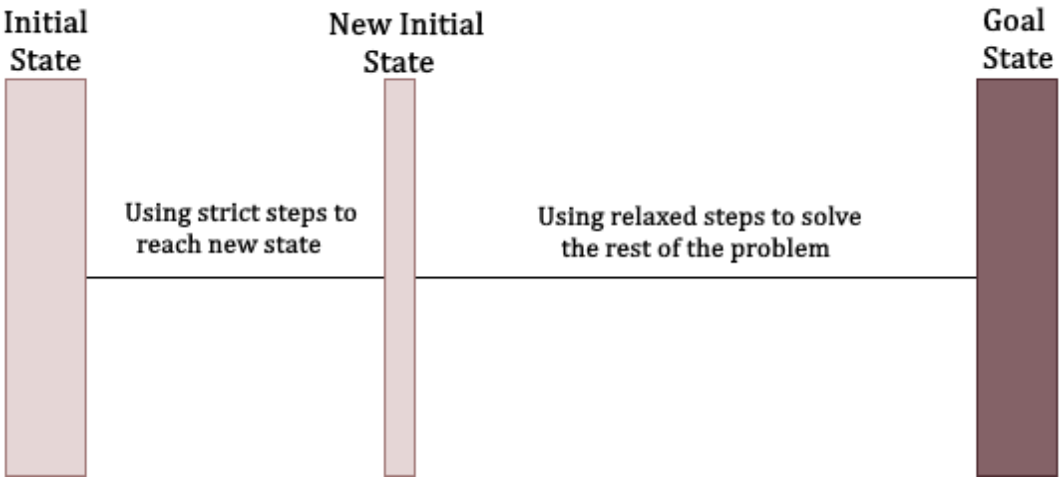
Σχήμα 3.1 Διαχωρισμός προβλήματος και μετακίνηση σε νέες καταστάσεις

Εάν ενώσουμε τις ακολουθίες από δράσεις που μας παίρνουν από μια κατάσταση σε άλλη, θα έχουμε τη λύση του προβλήματος σαν ολοκληρωμένο πλάνο. Οι ακολουθίες από δράσεις αυτές σε κάθε επανάληψη είναι ένα υπο-πλάνο (sub-plan). Η επίλυση ενός υπο-πλάνου σε κάθε επανάληψη χρησιμοποιεί δύο είδη βημάτων (steps): αυστηρά βήματα (strict steps) και χαλαρωμένα βήματα (relaxed steps). Στα αυστηρά βήματα όλοι οι περιορισμοί του αρχικού προβλήματος διατηρούνται και μια δράση δεν μπορεί να εκτελεστεί παράλληλα με μια άλλη αν υπάρχει παρέμβαση μεταξύ τους. Αντίθετα στα χαλαρωμένα βήματα επιτρέπεται να εκτελεστούν παράλληλα δράσεις που παρεμβάλλονται. Ένα παράδειγμα είναι ένα χέρι που μπορεί να μετακινεί ένα κύβο κάθε φορά να μπορεί σε χαλαρωμένη κατάσταση να κρατά παραπάνω από ένα κύβο.

Αυτό επιτυγχάνεται με αλλαγή περιορισμών στο meta encoding της plasp. Συγκεκριμένα αλλάζουμε τον κανόνα :- 1 < parallel, parallel < 4, active(A), not alright(A,t). σε :- 1 < parallel, parallel < 4, active(A), not alright(A,t), t<c1+1. Ο κανόνας με το κατηγόρημα alright(A,t) ελέγχει αν όλες οι ενεργές δράσεις δεν παρεμβάλλονται. Η σταθερά c1 με τον έλεγχο που προστέθηκε στο τέλος επιτρέπει την αγνόηση αυτού του περιορισμού για χρονική στιγμή μεγαλύτερη του c1.

Η σταθερά c1 συμβολίζει πόσα βήματα θα είναι αυστηρά βήματα. Άρα, λύνοντας το πρόβλημα με ένα αριθμό αυστηρών βημάτων θα πάρουμε μια νέα αρχική κατάσταση που είναι η κατάσταση στην οποία βρεθήκαμε μετά την επίλυση. Κρατούμε την λύση μέχρι αυτό το σημείο δημιουργώντας ένα νέο πρόβλημα με αρχική κατάσταση την κατάσταση στην οποία φτάσαμε και κατάσταση στόχου να παραμένει η ίδια. Ο κανόνας για τη νέα αρχική κατάσταση με τη λογική της σταθεράς c1 γράφεται

initialState(t,X,value(X,true)):-holds(X,value(X,true),t), t=c1.”. Το υπόλοιπο πρόβλημα μετά τα αυστηρά βήματα λύνεται με χαλαρωμένα βήματα. Αυτό φαίνεται στο Σχήμα 3.2.



Σχήμα 3.2 Χρήση αυστηρών και χαλαρωμένων βημάτων σε υπο-πλάνο

3.2.2 Είδη Ευρετικών

Για να προχωρήσουμε σε μια επόμενη κατάσταση που είναι πιο κοντά στην τελική κατάσταση του αρχικού προβλήματος χρειαζόμαστε ευρετικά. Επίσης χρειαζόμαστε ένα τρόπο για αποφυγή της δημιουργίας τοπικών προβλημάτων, επίλυσης τους και επαναδημιουργίας τους έτσι ώστε να μην έχουμε τον επιλυτή να μην προχωρά. Ακόμα θέλουμε να μειώσουμε τον αριθμό των χαλαρωμένων βημάτων αφού αγνοούν περιορισμούς του προβλήματος. Για να το επιτύχουμε αυτό και να μετρήσουμε την απόσταση μιας νέας κατάστασης με της κατάστασης στόχου γίνεται χρήση των ακόλουθων κανόνων:

achnew(X,K):- goal(X,V), holds(X,V,t),t=c1+K, num(K).
#minimize{-(24 - 3*K),X,K:achnew(X,K)}.

Με το κατηγόρημα achnew(X,K) αποθηκεύουμε τους στόχους που ισχύουν σε χαλαρωμένα βήματα. Το K στο κατηγόρημα μας δείχνει σε ποιο χαλαρωμένο βήμα βρισκόμαστε, για παράδειγμα achnew(X,1) είναι ένας στόχος που ισχύει στο πρώτο χαλαρωμένο βήμα. Ένας στόχος που επιτυγχάνεται σε ένα χαλαρωμένο βήμα θα ισχύει

και στα υπόλοιπα μετά. Για K = 0 δεν έχουμε χαλαρωμένο βήμα αλλά συμβολίζουμε τη νέα αρχική κατάσταση που βρήκαμε. Η απόσταση αυτής της κατάστασης με της κατάστασης στόχου βρίσκεται στον αριθμό των στόχων που ισχύουν στα χαλαρωμένα βήματα. Στη συνάρτηση ελαχιστοποίησης (μεγιστοποίησης αν την δούμε χωρίς το αρνητικό πρόσημο στην αρχή) προσπαθούμε ουσιαστικά να μεγιστοποιήσουμε τον αριθμό των στόχων που βρίσκονται στις χαλαρωμένες καταστάσεις με μεγαλύτερο βάρος να δίνεται αν ο στόχος επιτευχθεί σε μικρότερο χαλαρωμένο βήμα. Αυτό επιτυγχάνεται με την αφαίρεση από το 24 που είναι ο αριθμός βάρους στην κατάσταση που βρισκόμαστε. Όσο μεγαλύτερο είναι το χαλαρωμένο βήμα τόσο μικρότερο το βάρος. Συγκεκριμένα το βάρος μειώνεται κατά 3 για κάθε χαλαρωμένο βήμα.

Στην προσπάθεια επίλυσης του προβλήματος χρειαζόμαστε ευρετικά που θα μας καθοδηγήσουν προς τον στόχο. Ένα τέτοιο ευρετικό που προκύπτει από τον ορισμό της συνάρτησης ελαχιστοποίησης είναι το ευρετικό κόστους (cost heuristic). Η clingo με την επίλυση της συνάρτησης επιστρέφει τα κόστη των λύσεων που βρήκε, από τα οποία παίρνουμε το πιο βελτιστοποιημένο (optimized). Το ευρετικό κόστους στο σύστημα δουλεύει με την επίλυση πρώτα του προβλήματος μόνο με χαλαρωμένα βήματα. Μετά γίνεται προσπάθεια επίλυσης με ένα αριθμό αυστηρών βημάτων. Εάν το κόστος της δεύτερης προσπάθειας με τα αυστηρά βήματα ήταν μικρότερο από της πρώτης με χαλαρωμένα βήματα μόνο, τότε θα φυλάξουμε τη νέα αρχική κατάσταση μαζί με το κόστος της δεύτερης προσπάθειας. Αν δεν το επιτύχουμε, τότε θα αυξάνουμε τον αριθμό των αυστηρών βημάτων μέχρι να ισχύει αυτή η συνθήκη. Με τη νέα αρχική κατάσταση επαναλαμβάνουμε αυτή τη διαδικασία μέχρι να φτάσουμε σε λύση του αρχικού προβλήματος.

Το δεύτερο ευρετικό που χρησιμοποιείται είναι τα χαλαρωμένα βήματα, με συνθήκη σύγκρισης των χαλαρωμένων βημάτων που χρησιμοποιήθηκαν σε κάθε επανάληψη. Αν μειώθηκαν, τότε θεωρούμε ότι προχωρούμε προς την λύση. Η απόδοση αυτού του ευρετικού είναι διαφορετική από το ευρετικό κόστους, με αποτέλεσμα διαφορετικά ευρετικά να είναι πιο χρήσιμα ανάλογα με το είδος του προβλήματος.

Κεφάλαιο 4

Επεκτάσεις Ευρετικών Μεθόδων στο Σύστημα Plasp

4.1 Επέκταση της Παραμέτρου Χαλαρότητας Καταστάσεων

Η μέθοδος χαλάρωσης στο σύστημα που χρησιμοποιούμε στα πλαίσια της εργασίας αυτής επιτρέπει τη δημιουργία καταστάσεων από την παράλληλη εκτέλεση δράσεων που φυσιολογικά δεν μπορούν να εκτελεστούν παράλληλα γιατί είναι αμοιβαία αποκλειόμενες. Με τη χαλάρωση αυτή επιτρέπεται η δημιουργία καταστάσεων με γεγονότα τα οποία σε κανονικές συνθήκες δεν μπορούν να εμφανίζονται ταυτόχρονα. Το μέγεθος της χαλάρωσης αυτής προσδιορίζεται από την παράμετρο χαλαρότητας καταστάσεων, που καθορίζεται από τον χρήστη. Έτσι για παράδειγμα αν η παράμετρος αυτή έχει την τιμή 3 στον κόσμο των κύβων, τότε επιτρέπεται η παράλληλη εκτέλεση 3 μετακινήσεων του ιδίου κύβου σε 3 διαφορετικά μέρη. Αυτό έχει ως αποτέλεσμα τη δημιουργία χαλαρωμένων καταστάσεων όπου ένας κύβος μπορεί να βρίσκεται σε 3 διαφορετικά μέρη.

Στο παρόν σύστημα η παράμετρος αυτή παίρνει μια τιμή που δίνεται σε αρχείο παραμέτρων (options file) και διατηρεί αυτή την τιμή καθ' όλη τη διάρκεια της εκτέλεσης. Μια αλλαγή αυτής της απλής λειτουργίας ήταν να μεταβάλλουμε τη τιμή της παραμέτρου χαλαρότητας κατά την διάρκεια της εκτέλεσης, για να δούμε αν

συνεισφέρει στην επίλυση να περιορίζουμε την τιμή του καθώς το πρόγραμμα προχωρά σε αριθμό επαναλήψεων.

Στο αρχείο παραμέτρων του συστήματος το όνομα της παραμέτρου χαλαρότητας είναι “k”. Για να υλοποιήσουμε αυτή τη λειτουργία, θα προσθέσουμε στο αρχείο την επιλογή “kReduction”, που υποδηλώνει μετά από πόσες επαναλήψεις θα γίνει μείωση της παραμέτρου χαλαρότητας. Η μείωση θα γίνεται μέχρι τον αριθμό 2, αφού όπως εξηγήθηκε με τη τιμή 1 δεν θα έχουμε χαλάρωση. Για kReduction ίσο του 0 (προκαθορισμένη τιμή) δεν γίνεται μείωση της παραμέτρου χαλαρότητας και το σύστημα λειτουργεί όπως χωρίς την αλλαγή. Αλλιώς θα περιορίζεται κατά 1 κάθε φορά που φτάνουμε στον αριθμό επαναλήψεων που δίνεται με την παράμετρο.

Ας δούμε ένα παράδειγμα εκτέλεσης (μόνο κομμάτι με επαναλήψεις) με μηνύματα να τυπώνονται όταν αλλάζει η παράμετρος χαλαρότητας. Θα χρησιμοποιήσουμε k = 5 και kReduction = 2.

While problem isn't solved trying strict runs...
(iteration:1 / steps:3)
(iteration:2 / steps:3)
Changed k to 4
(iteration:3 / steps:3)
(iteration:4 / steps:3)
Changed k to 3
(iteration:5 / steps:3)
(iteration:6 / steps:3)
Changed k to 2
(iteration:7 / steps:3)
(iteration:8 / steps:3)
(iteration:9 / steps:3)
Done.

Όπως παρατηρούμε, μετά την ολοκλήρωση 2 επαναλήψεων, η παράμετρος χαλαρότητας περιορίζεται μέχρι να φτάσει την τιμή 2.

Ένα ακόμα option που υλοποιήθηκε για επέκταση της παραμέτρου χαλαρότητας είναι το “kReductionDirection”, το οποίο υποδηλώνει πως μετράμε την αύξηση των επαναλήψεων για αλλαγή της παραμέτρου. Εάν οι επαναλήψεις αυξάνονταν πάντα προς μεγαλύτερους αριθμούς, τότε δεν θα υπήρχε ανάγκη για να ξεχωρίσουμε περιπτώσεις. Όμως, στο πρόγραμμα μας κάποτε χρειάζονται να γίνουν οπισθοδρομήσεις στις επαναλήψεις. Εάν kReductionDirection = 0 (προκαθορισμένη τιμή), τότε η παράμετρος χαλαρότητας αλλάζει μόνο αν η επανάληψη αυξηθεί προς μεγαλύτερο αριθμό από τον μέγιστο που έχει φτάσει για kReduction φορές. Εάν kReductionDirection = 1, τότε μετράμε και τις οπισθοδρομήσεις στις επαναλήψεις ως πρόοδο των επαναλήψεων. Άρα, η παράμετρος χαλαρότητας αλλάζει γενικά με αλλαγή μια επανάληψης σε άλλη επανάληψη (μπροστά ή πίσω) για kReduction φορές. Εάν το kReduction είναι 0, τότε δεν αλλάζει κάτι αυτή την επιλογή.

4.2 Επέκταση της Παραμέτρου Επανεκκίνησης

Κάποιες φορές κατά την εκτέλεση ο επιλυτής μπορεί να βρεθεί σε κατάσταση αδιέξοδο, από την οποία να μην υπάρχει πλάνο για την τελική κατάσταση και να μην βρει λύση. Αυτό συμβαίνει για αυξανόμενο αριθμό αυστηρών βημάτων, επειδή δεν μπορεί να βρει λύση που να ικανοποιεί τις συνθήκες που δίνουμε (πχ με καλύτερο κόστος) ή επειδή δεν μπορεί να βρει λύση γενικά.

Για αντιμετώπιση αυτού του προβλήματος, υλοποιήθηκε στο σύστημα η παράμετρος επανεκκίνησης που ονομάζεται “reset”. Ο αριθμός της παραμέτρου υποδηλώνει πόσες φορές ο επιλυτής μπορεί να βρει το πρόβλημα μη-ικανοποιήσιμο (επιστρέφοντας “unsatisfied” και κόστος ίσο με 0) πριν κάνουμε οπισθοδρόμηση (reset δηλαδή) στην επανάληψη 1, για να ξαναπροσπαθήσει από την αρχή. Εάν η παράμετρος επανεκκίνησης έχει τιμή 0 στο αρχείο παραμέτρων, τότε η λειτουργία αυτή δεν χρησιμοποιείται. Όταν γίνεται οπισθοδρόμηση, το πρόγραμμα επιστρέφει στη κατάσταση του κόσμου στην οποία βρισκόταν την τελευταία φορά που έφτασε στην επανάληψη στην οποία επιστρέφει.

Για παράδειγμα, μια εκτέλεση που χωρίς την παράμετρο επανεκκίνησης δεν μπορεί να προχωρήσει μετά την τρίτη επανάληψη:

While problem isn't solved trying strict runs...
(iteration:1 / steps:3)
(iteration:2 / steps:3)
(iteration:3 / steps:3)
(iteration:3 / steps:5)
(iteration:3 / steps:7)
…
(iteration:3 / steps:27)
(iteration:3 / steps:29)
…

Συγκριτικά, με reset = 3 η εκτέλεση μπορεί να προχωρήσει και τελικά βρίσκει λύση στο πρόβλημα:

(iteration:1 / steps:3)
(iteration:2 / steps:3)
(iteration:3 / steps:3)
Reset = 1
(iteration:3 / steps:5)
Reset = 2
(iteration:3 / steps:7)
Reset = 3
UPDATE
Reset to iteration 1
(iteration:1 / steps:3)
(iteration:2 / steps:3)
(iteration:3 / steps:3)
(iteration:3 / steps:5)
Done.

Η πρώτη σκέψη για επέκταση αυτής της παραμέτρου ήταν να επιστρέφουμε πίσω σε κάποια άλλη επανάληψη αντί της πρώτης κάθε φορά, αφού σε κάποια προβλήματα μπορεί να είναι χρονοβόρο να ξεκινούμε ξανά από την αρχή κάθε φορά που ο επιλυτής

δεν μπορεί να προχωρήσει. Αυτό όμως μπορεί επίσης να είναι πιο χρονοβόρο τελικά, αφού η λάθος επιλογή στο μονοπάτι της λύσης μπορεί να μην έγινε στις τελευταίες επαναλήψεις αλλά πιο πίσω. Ο επιλυτής έτσι τελικά θα ξαναβρεθεί σε αδιέξοδο μετά την οπισθοδρόμηση. Δηλαδή, σε κάποιες περιπτώσεις ίσως να είναι καλύτερο να επιστρέφει πίσω στην πρώτη επανάληψη αντί σε κοντινές προηγούμενες.

Εκτός από αυτό το πρόβλημα όμως, παρατηρήσαμε ότι υπάρχουν περιπτώσεις τις οποίες δεν μπορεί να ανιχνεύσει η παράμετρος επανεκκίνησης, όπου ο επιλυτής βρίσκεται σε αδιέξοδο αλλά δεν επιστρέφει unsatisfied συνεχόμενα. Ο επιλυτής επιστρέφει κανονικά το κόστος που βρίσκει που μπορεί να μην είναι ίσο με 0. Αν το κόστος δεν είναι 0, δεν ενεργοποιείται η παράμετρος επανεκκίνησης, άρα σε αυτές τις περιπτώσεις δεν μπορούμε να προχωρήσουμε στην εκτέλεση ούτε με την απλή λειτουργία που προσφέρει.

Για παράδειγμα έχουμε μια εκτέλεση όπου τυπώνονται τα κόστη στο κάθε επανάληψη:

While problem isn't solved trying strict runs...
(iteration:1 / steps:3)
Cost = -21
(iteration:1 / steps:5)
Cost = -21
(iteration:1 / steps:7)
Cost = -21
(iteration:1 / steps:9)
Cost = -21
(iteration:1 / steps:11)
Cost = -165
(iteration:2 / steps:3)
Cost = -252
(iteration:3 / steps:3)
Cost = -252
(iteration:3 / steps:5)
Cost = -252

(iteration:3 / steps:7)
Cost = -252
(iteration:3 / steps:9)
Cost = -252
(iteration:3 / steps:11)
Cost = -252
…
(iteration:3 / steps:27)
Cost = -252
(iteration:3 / steps:29)
Cost = -252

Όπως φαίνεται από αυτό το παράδειγμα, κάποιες φορές, όπως στην πρώτη επανάληψη, ο επιλυτής επιστρέφει το ίδιο κόστος αλλά τελικά καταφέρνει να προχωρήσει, ενώ σε άλλες δεν μπορεί, όπως στην επανάληψη 3.

Σε αυτές τις περιπτώσεις θέλουμε μια επέκταση της παραμέτρου επανεκκίνησης ή μια νέα παράμετρο που να δίνει τη λειτουργία της οπισθοδρόμησης όταν παρατηρούνται τέτοια προβλήματα, χωρίς απαραίτητα να πηγαίνουμε πίσω στην πρώτη επανάληψη, αφού όπως είπαμε μπορεί να είναι άσκοπα χρονοβόρα μια τέτοια επιλογή. Στην εργασία αυτή έγινε η επιλογή να μείνει η παράμετρος επανεκκίνησης όπως είναι ήδη υλοποιημένη έτσι ώστε να μπορεί να χρησιμοποιείται, αφού είναι αποτελεσματική σε κάποιες περιπτώσεις με την υλοποίηση που ήδη έχει. Ως επέκταση της παραμέτρου επανεκκίνησης, έχει υλοποιηθεί μια νέα παράμετρος την οποία θα δούμε στη συνέχεια, η οποία έχει ως σκοπό να αντιμετωπίσει τέτοια προβλήματα διαφορετικά από την παράμετρο επανεκκίνησης και να εφαρμόζει στρατηγικές για οπισθοδρόμηση.

4.2.1 Υλοποίηση της Παραμέτρου Επανάληψης

Η παράμετρος που υλοποιήθηκε ως επέκταση της παραμέτρου επανεκκίνησης είναι η παράμετρος επανάληψης “repeat”. Η παράμετρος αυτή αποσκοπεί στην οπισθοδρόμηση σε προηγούμενες επαναλήψεις όταν ο επιλυτής βρεθεί σε αδιέξοδο. Πιο συγκεκριμένα, η παράμετρος επανάληψης μετράει πόσες φορές ο επιλυτής έχει βρει την ίδια λύση

παρατηρώντας τα κόστη. Όταν ο επιλυτής μας δώσει repeat φορές μια ίδια λύση, τότε γίνεται οπισθοδρόμηση. Εάν η παράμετρος επανάληψης είναι 0 στο αρχείο παραμέτρων, τότε δεν θα χρησιμοποιηθεί η λειτουργία της. Με μεγαλύτερη τιμή από 0 δηλώνουμε τον αριθμό των επαναλήψεων στις λύσεις που μπορεί να επιστρέψει ο επιλυτής πριν γίνει οπισθοδρόμηση.

Η στρατηγική που επιλέγει η παράμετρος επανάληψης για την οπισθοδρόμηση υποδηλώνεται από μια επιπλέον παράμετρο που υλοποιήθηκε με όνομα “backtrackAggression”. Η παράμετρος αυτή επιτρέπει 3 είδη χειρισμού της οπισθοδρόμησης. Για backtrackAggression = 0 επιστρέφουμε πίσω στην προηγούμενη επανάληψη μείον ενός μετρητή, του οποίου η λειτουργία θα εξηγηθεί αργότερα. Για backtrackAggression = 1 θέλουμε μια πιο δραστική οπισθοδρόμηση που θα επιστρέφει στην επανάληψη i/2 μείον του μετρητή. Για backtrackAggression = 2 θέσαμε την πιο δραστική επιλογή που είναι οπισθοδρόμηση στην επανάληψη 1 και κάθε 2 φορές που επιστρέφουμε πίσω στην επανάληψη 1 θα μειώνουμε την τιμή του παραμέτρου χαλαρότητας κατά 1, εκτός αν έχει ήδη φτάσει στη ελάχιστη της τιμή ίση με 2. Ο περιορισμός της παραμέτρου χαλαρότητας προέρχεται από την ιδέα που αφορούσε αυτή την παράμετρο που αναφέρθηκε προηγουμένως. Μετά από κάθε οπισθοδρόμηση η τιμή των επαναλήψεων λύσεων που βρέθηκαν γίνεται 0, ξαναξεκινώντας το μέτρημα για επαναλήψεις.

Ο λόγος που προσθέσαμε τον μετρητή στις πρώτες δύο επιλογές στρατηγικής οπισθοδρόμησης, είναι επειδή σε περιπτώσεις που βρίσκουμε αδιέξοδο σε μια επανάληψη και επιστρέφουμε, θα μπορούσαμε να προχωρήσουμε ξανά στην ίδια επανάληψη και να οπισθοδρομήσουμε στην ίδια, μπαίνοντας σε κύκλο. Για να αντιμετωπίσουμε αυτά τα σενάρια, με τον μετρητή θα επιστρέφουμε ουσιαστικά για κάθε επιπλέον οπισθοδρόμηση ακόμα μια επανάληψη πιο πίσω. Έτσι, δεν θα επιστρέφουμε δεύτερη φορά στην ίδια επανάληψη σε αυτές τις περιπτώσεις. Κάθε φορά που φτάνουμε στον αριθμό των επιτρεπόμενων επαναλήψεων λύσεων, αυξάνουμε αυτό τον μετρητή που φυλάει πόσες φορές αποτύχαμε. Το μετρητής αυτός γίνεται ξανά 0 όταν έχουμε τον μέγιστο αριθμό επαναλήψεων λύσεων που τέθηκαν, αλλά η επανάληψη που φτάσαμε είναι πιο μεγάλη από τη μέγιστη που φτάσαμε, έτσι ώστε να μην στέλνει το πρόγραμμα πίσω πολλές επαναλήψεις εάν έχει προχωρήσει μπροστά. Ο

μετρητής επίσης βοηθά στην ιδέα του να μην καθυστερούμε σε αχρείαστες οπισθοδρομήσεις αν συνεχόμενα παγιδευόμαστε σε αδιέξοδα στην εκτέλεση, που είναι ακόμα ένα πρόβλημα όσο αφορά την επιλογή στρατηγικής οπισθοδρόμησης, όπως είχε αναφερθεί προηγουμένως με τα προβλήματα της παραμέτρου επανεκκίνησης.

Ας δούμε ένα παράδειγμα χρήσης της παραμέτρου επανάληψης με επιλογές repeat = 3, backtrackAggression = 0 και repeatPerIteration = 0:

While problem isn't solved trying strict runs...
(iteration:1 / steps:3)
(iteration:2 / steps:3)
(iteration:3 / steps:3)
(iteration:4 / steps:3)
(iteration:5 / steps:3)
(iteration:5 / steps:5)
Repeat = 1
(iteration:5 / steps:7)
Repeat = 2
(iteration:5 / steps:9)
Repeat = 3
UPDATE
Reset to iteration 4
(iteration:4 / steps:3)
Repeat = 1
(iteration:5 / steps:3)
Repeat = 2
(iteration:5 / steps:5)
Repeat = 3
UPDATE
Reset to iteration 3
(iteration:3 / steps:3)
Repeat = 1
(iteration:4 / steps:3)

Repeat = 2
(iteration:5 / steps:3)
Repeat = 3
UPDATE
Reset to iteration 2
(iteration:2 / steps:3)
(iteration:3 / steps:3)
(iteration:4 / steps:3)
(iteration:4 / steps:5)
(iteration:5 / steps:3)
(iteration:6 / steps:3)
(iteration:7 / steps:3)
(iteration:8 / steps:3)
(iteration:9 / steps:3)
(iteration:10 / steps:3)
(iteration:10 / steps:5)
(iteration:11 / steps:3)
(iteration:12 / steps:3)
Done.

Παρατηρούμε ότι στο πρόγραμμα επιστέψαμε πίσω σε προηγούμενη επανάληψη 3 φορές. Την πρώτη φορά που βρεθήκαμε σε αδιέξοδο στην επανάληψη 5, επιστρέψαμε πίσω στην επανάληψη 4. Όταν ξαναβρεθήκαμε σε αδιέξοδο στην επανάληψη 5 για δεύτερη φορά, επιστρέψαμε στην επανάληψη 3 λόγω του μετρητή. Την τρίτη φορά που πάλι βρεθήκαμε σε αδιέξοδο στην επανάληψη 5, επιστρέψαμε στην επανάληψη 2, που τελικά μας έβαλε στο σωστό μονοπάτι για επίλυση του προβλήματος. Όπως παρατηρούμε εδώ, αφού repeatPerIteration = 0, τότε η ιδέα της επανάληψης λύσεων διατηρείται μεταξύ των επαναλήψεων. Έτσι, τη δεύτερη και τρίτη φορά που γίνεται η οπισθοδρόμηση, παρατηρούμε ότι είχαμε μετρήσει επανάληψη λύσεων από προηγούμενες επαναλήψεις, που τελικά προκαλεί οπισθοδρόμηση την τρίτη φορά που παρατηρείται, αφού η παράμετρος επανάληψης ισούται με 3.

Εκτός από αυτές τις λειτουργίες της παραμέτρου επανάληψης, αποφασίσαμε να προσθέσουμε και ακόμα κάποιες επιπλέον λειτουργίες, για να αντιμετωπίσουμε συγκεκριμένα σενάρια που μπορεί να μην αφήνουν το πρόγραμμα να προχωρήσει. Το πρώτο σενάριο είναι στις περιπτώσεις που η παράμετρος επανάληψης δεν αφήνει το πρόγραμμα να προχωρήσει, επειδή έχει μαζέψει πολλές λύσεις που θεωρούνται επαναλαμβανόμενες αν ξαναεμφανιστούν, αλλά επίσης έστρεψε το πρόγραμμα σε αρκετά πίσω αριθμό επανάληψης. Λόγω της επανάληψης αυτής στις λύσεις, δεν θα ξαναμπορέσει να προχωρήσει σε σημείο που να μπορεί να συνεχίσει την επίλυση, αφού η παράμετρος επανάληψης θα το στέλνει συνεχόμενα πίσω. Σε αυτή την περίπτωση, θα χρησιμοποιείται η παράμετρος ανεκτικότητας στην οπισθοδρόμηση με όνομα “backtrackTolerance”, που θα αυξάνει την τιμή των επιτρεπόμενων επαναλήψεων λύσεων κατά τον αριθμό που ορίζεται, κάθε φορά που επιστρέφουμε πίσω στην επανάληψη 1. Άρα, θα γίνεται πιο ανεκτική η παράμετρος επανάληψης κάθε φορά που επιστρέφει στην αρχή, για να δώσει ευκαιρία στον επιλυτή να ξαναδοκιμάσει. Εάν η παράμετρος αυτή έχει τιμή 0, τότε ουσιαστικά δεν θα χρησιμοποιείται.

Στις περιπτώσεις που χρησιμοποιούμε ευρετικό χαλαρωμένων βημάτων, παρατηρήθηκε ότι μπορεί να έχουμε πολλές φορές την ίδια λύση μέχρι να προχωρήσουμε τελικά σε επόμενη επανάληψη. Σε αυτές τις περιπτώσεις είναι χρήσιμο ο αριθμός των επαναλήψεων λύσεων που κρατά η παράμετρος επανάληψης να μην διατηρείται σε επόμενο αριθμό επανάληψης και να μηδενίζεται, αφού αλλάζουμε αριθμό επανάληψης. Έτσι, θα επιτρέπει στο πρόγραμμα να προχωρήσει πιο εύκολα, ειδικά αν η τιμή που βάλαμε στην παράμετρο επανάληψης είναι μικρή. Η παράμετρος που το επιτρέπει αυτό είναι η παράμετρος ρύθμισης της παραμέτρου επανάληψης ανά αριθμό επανάληψης με όνομα “repeatPerIteration”. Αν η παράμερος αυτή έχει την τιμή 1, ο αριθμός των επαναλήψεων λύσεων που βρήκε η παράμετρος επανάληψης γίνεται 0 σε κάθε επόμενη επανάληψη αντί να διατηρείται. Αλλιώς, αν η παράμετρος αυτή έχει την τιμή 0, τότε ο αριθμός των επαναλήψεων λύσεων που βρήκε η παράμετρος επανάληψης διατηρείται μεταξύ επαναλήψεων. Εάν η παράμετρος επανάληψης διατηρεί την αριθμό αυτό, τότε θεωρούμε ότι η ιδέα της επανάληψης λύσεων μπορεί να ξεκίνησε από προηγούμενες επαναλήψεις εκτός από μόνο την τρέχον. Δηλαδή δίνουμε παραπάνω σημασία στο γεγονός ότι μπορεί να είμαστε σε λάθος μονοπάτι για παραπάνω από μια επανάληψη στη λειτουργία της παραμέτρου επανάληψης.

Αξίζει να σημειωθεί ότι η παράμετρος επανάληψης καλύπτει και τη λειτουργία της παραμέτρου επανεκκίνησης στις περιπτώσεις που το πρόγραμμα επιστρέψει unsatisfied, αφού θα παίρνει συνεχόμενα τιμές κόστους 0. Η αντιμετώπιση όμως της παραμέτρου επανάληψης όπως είδαμε είναι διαφορετική και επηρεάζεται και από τις υπόλοιπες παραμέτρους που υλοποιήθηκαν.

4.2.2 Επέκταση Παραμέτρου Επανάληψης με το Σύστημα TorchLight

Επειδή η υλοποίηση της παραμέτρου επανάληψης δεν αποτελεί ολική λύση του πρόβληματος που είχαμε αλλά ένα ευρετικό που μας βοηθά στην επίλυση, αποφασίσαμε να χρησιμοποιήσουμε και το σύστημα TorchLight που αναλύει τον χώρο αναζήτησης για να δώσει στατιστικά για πιθανή λύση ως επιπλέον ευρετικό. Το σύστημα αυτό βοηθά να δούμε αν υπάρχουν αδιέξοδα (dead ends) στο πρόβλημα που προσπαθούμε να λύσουμε, δοκιμάζοντας να εκτελέσει τυχαίες δράσεις και παρατηρώντας τις καταστάσεις τις οποίες βρίσκει [7].

Για το TorchLight χρησιμοποιούμε την παράμετρο -s <num> που ρυθμίζει τον αριθμό των καταστάσεων που θα δοκιμαστούν (samples) και την παράμετρο -d <num> που ρυθμίζει το βάθος των καταστάσεων επιλογής [7]. Αυτές οι παράμετροι μπορούν να ρυθμιστούν από το αρχείο παραμέτρων με τις τιμές των παραμέτρων “torchlightS” και “torchlightD” που έχουν υλοποιηθεί. Οι τιμές που παίρνουν αυτές οι παράμετροι θα εξηγηθούν στη συνέχεια αφού δούμε τη λειτουργία του TorchLight. Το TorchLight με τις δύο αυτές παραμέτρους θα δώσει μια απάντηση ποσοστού για την ανάλυση, που μπορούμε να χρησιμοποιήσουμε για να δούμε κατά πόσο η κατάσταση στην οποία βρισκόμαστε σε μια στιγμή μπορεί να φτάσει σε λύση συνεχίζοντας. Επίσης, δίνουμε την παράμετρο -E για να παίρνουμε μια επιπλέον τιμή με βάση την ανάλυση Enforced Hill-Climbing (EHC) από το TorchLight [7], η οποία διαφέρει σε κάποιες περιπτώσεις από την άλλη τιμή, άρα μπορεί να χρησιμοποιηθεί ως επιπλέον πληροφορία.

Για τις απαντήσεις του TorchLight παρατηρήθηκε ότι αν η προσέγγιση που επιστρέφει είναι 0%, τότε πολλές φορές το σύστημα δεν θα προχωρήσει σε λύση επειδή βρίσκει σχεδόν 100% αδιέξοδα. Άρα, είναι χρήσιμο να έχουμε μια παράμετρο όπως την παράμετρο επανάληψης, αλλά για το πόσες φορές μπορούμε να πάρουμε 0% από το

TorchLight πριν κάνουμε οπισθοδρόμηση. Άρα, αν το κανονικό αποτέλεσμα του TorchLight είναι 0% ή το αποτέλεσμα του EHC είναι 0%, τότε το μετράμε σαν αποτυχία (παρόμοιο με επανάληψη στην παράμετρο επανάληψης). Οι υπόλοιπες τιμές δεν φάνηκαν ιδιαίτερα χρήσιμες από δοκιμές, επειδή είναι γενικά κοντινές μεταξύ τους στα πλείστα προβλήματα.

Η παράμετρος που υλοποιεί αυτή τη λειτουργία είναι η παράμετρος αποτυχιών στην αναζήτηση του TorchLight με όνομα “torchlightFails”. Αν η παράμετρος είναι 0, τότε απενεργοποιείται η χρήση του TorchLight. Αλλιώς, αν πάρουμε τιμές 0% ίσες με τον αριθμό που δίνουμε στην παράμετρο, τότε θα έχουμε οπισθοδρόμηση. Ο αριθμός των προσεγγίσεων ίσων με 0% που βρήκε το TorchLight γίνεται 0 για κάθε επανάληψη, αφού δεν φάνηκε ιδιαίτερα χρήσιμο να κρατούμε την ιδέα της αποτυχίας για επόμενες επαναλήψεις για τα αποτελέσματα του TorchLight. Ο λόγος αυτής της επιλογής είναι ότι αν το πρόγραμμα κατάφερε να προχωρήσει στην εκτέλεση, τότε η προηγούμενη προσέγγιση 0% που μπορεί να είχαμε πάρει από το TorchLight δεν ήταν ακριβής. Έτσι, μπορούμε να επικεντρωθούμε στο αν έχουμε βρεθεί σε αδιέξοδο στην τρέχον επανάληψη, αντί κατά πόσο έχουμε μπει σε λάθος μονοπάτι από προηγούμενες επαναλήψεις με τις απαντήσεις του TorchLight.

Η οπισθοδρόμηση της παραμέτρου αποτυχιών του TorchLight, όπως και στην παράμετρο επανάληψης, θα ακολουθεί την παράμετρο επιλογής στρατηγικής οπισθοδρόμησης (backtrackAggression). Η λειτουργία του μετρητή παραμένει η ίδια και μπορεί να χρησιμοποιηθεί τόσο από την παράμετρο επανάληψης, όσο και από την παράμετρο αποτυχιών του TorchLight μαζί. Το ίδιο συμβαίνει για την παράμετρο ανεκτικότητας (backtrackTolerance), δηλαδή εάν η παράμετρος αποτυχιών του TorchLight ήταν η αιτία να πάμε στην πρώτη επανάληψη, τότε θα αυξάνουμε τη μέγιστη τιμή των επιτρεπτών απαντήσεων 0% πριν γίνει οπισθοδρόμηση. Αλλιώς, εάν ήταν η παράμετρος επανάληψης, θα αυξάνουμε την επιτρεπτή τιμή επαναλήψεων λύσεων αυτής της παραμέτρου. Αυτό ισχύει φυσικά εάν και οι δύο παράμετροι είναι ενεργοποιημένες ταυτόχρονα. Αν γίνει οπισθοδρόμηση από οποιαδήποτε από αυτές τις παραμέτρους, τότε οι τιμές που διατηρούν τόσο αυτή όσο και η άλλη παράμετρος αντίστοιχα σχετικά με τις απαντήσεις θα γίνουν 0, για να αποφύγουμε

επαναλαμβανόμενες αχρείαστες οπισθοδρομήσεις, αφού μόλις θα είχε γίνει μια οπισθοδρόμηση.

Όσον αφορά τις παραμέτρους αριθμού καταστάσεων επιλογής και βάθους των καταστάσεων επιλογής που αναφέρθηκαν προηγουμένως, παρατηρήθηκε ότι για εύρος τιμών 500 με 1500 στην τιμή του αριθμού καταστάσεων επιλογής που δηλώνει το torchlightS, το TorchLight επιστρέφει προσέγγιση 0% στις περιπτώσεις που φαίνεται ο επιλυτής να έχει βρεθεί σε αδιέξοδο. Αυτό είναι βοηθητικό για εμάς, άρα θέσαμε την προκαθορισμένη τιμή του σε 1000 στο πρόγραμμα. Όσο αφορά την τιμή του βάθους των καταστάσεων επιλογής που δηλώνει το torchlightD, δώσαμε μια τιμή βάθους που να είναι ικανοποιητική για την αναζήτηση, η οποία είναι 15, αφού παρατηρήθηκε ότι αριθμός μεγαλύτερος του 10 γενικά ήταν ικανοποιητικός.

Ας δούμε ένα παράδειγμα χρήσης της παραμέτρου αποτυχιών στην αναζήτηση του TorchLight με επιλογές torchlightFails = 3, backtrackAggression = 0, torchlightS = 1000 και torchlightD = 15:

While problem isn't solved trying strict runs...
(iteration:1 / steps:3)
(iteration:2 / steps:3)
(iteration:3 / steps:3)
(iteration:4 / steps:3)
(iteration:5 / steps:3)
(iteration:6 / steps:3)
(iteration:6 / steps:5)
TorchlightFails = 1
(iteration:6 / steps:7)
TorchlightFails = 2
(iteration:6 / steps:9)
TorchlightFails = 3
UPDATE
Reset to iteration 5
(iteration:5 / steps:3)

(iteration:5 / steps:5)
(iteration:6 / steps:3)
TorchlightFails = 1
(iteration:6 / steps:5)
TorchlightFails = 2
(iteration:6 / steps:7)
TorchlightFails = 3
UPDATE
Reset to iteration 4
(iteration:4 / steps:3)
(iteration:5 / steps:3)
(iteration:6 / steps:3)
(iteration:7 / steps:3)
(iteration:8 / steps:3)
(iteration:9 / steps:3)
(iteration:10 / steps:3)
(iteration:10 / steps:5)
(iteration:11 / steps:3)
(iteration:12 / steps:3)
Done.

Το πεδίο εφαρμογής και πρόβλημα που χρησιμοποιήθηκε ήταν το ίδιο με αυτό που χρησιμοποιήθηκε στο προηγούμενο παράδειγμα με την παράμετρο επανάληψης. Όπως παρατηρούμε, η οπισθοδρόμηση εδώ ήταν διαφορετική. Την πρώτη φορά το πρόγραμμα παγιδεύεται στην επανάληψη 6 και επιστρέφει στην επανάληψη 5. Τη δεύτερη φορά παγιδεύεται στην επανάληψη 6 ξανά, αλλά αυτή τη φορά επιστρέφει στην επανάληψη 4 λόγω του μετρητή που εξηγήσαμε προηγουμένως. Μετά την τελευταία οπισθοδρόμηση, το πρόγραμμα συνεχίζει κανονικά και βρίσκει λύση. Εδώ φαίνεται η διαφορά με την παράμετρο επανάληψης στο ότι η παράμετρος αποτυχιών στην αναζήτηση του TorchLight παρατηρεί για απαντήσεις 0% μόνο στην τρέχον επανάληψη, αφού χάνει τον αριθμό των τιμών 0% που παρατήρησε σε προηγούμενες επαναλήψεις αν το πρόγραμμα καταφέρει να προχωρήσει, όπως αναφέρθηκε και προηγουμένως.

Περαιτέρω αποτελέσματα χρήσης των παραμέτρων που υλοποιήθηκαν με διάφορους συνδυασμούς τους για εξέταση της επίδοσης τους θα ακολουθήσουν στην πειραματική μελέτη.

4.3 Σύνοψη Νέων Παραμέτρων

Όπως προαναφέρθηκε, το πρόγραμμα χρησιμοποιεί παραμέτρους από ένα αρχείο παραμέτρων. Στο αρχείο αυτό κάθε παράμετρος γράφεται με τη μορφή παράμετρος: τιμή (argument: value). Εάν μια παράμετρος δεν εμφανίζεται στο αρχείο, τότε θα δέχεται την προκαθορισμένη τιμή της. Το ίδιο συμβαίνει εάν η τιμή που δίνεται δεν συμπίπτει με τις αναμενόμενες τιμές χρήσης. Ακολουθεί μια σύνοψη των νέων υλοποιημένων παραμέτρων που έχουν περιγραφτεί προηγουμένως.

kReduction: Υποδηλώνει κάθε πόσες επαναλήψεις θα γίνει μείωση της παραμέτρου χαλαρότητας k. Για τιμή 0 (προκαθορισμένη τιμή) δεν γίνεται μείωση της παραμέτρου χαλαρότητας, δηλαδή η παράμετρος θα μένει σταθερή. Για επιλογή τιμών n > 0 η παράμετρος χαλαρότητας περιορίζεται κατά 1 μετά από τον αριθμό των επαναλήψεων που δίνεται, μέχρι να φτάσει την τιμή 2.

kReductionDirection: Χρησιμοποιείται για να ρυθμίσει τον τρόπο που μετρά τις επαναλήψεις η παράμετρος kReduction για τον περιορισμό της παραμέτρου χαλαρότητας. Για τιμή 0 (προκαθορισμένη τιμή) η αλλαγή της παραμέτρου γίνεται μόνο αν προχωρήσουμε σε επανάληψη με μεγαλύτερο αριθμό από τη μέγιστη επανάληψη που έχουμε φτάσει. Για τιμή 1 η τιμή της αλλάζει γενικά με οποιαδήποτε αλλαγή της επανάληψης, είτε προς μικρότερο είτε προς μεγαλύτερο αριθμό.

repeat: Μετρά πόσες φορές το πρόγραμμα επαναλαμβάνει λύσεις που έχει ήδη βρει σε επαναλήψεις με χρήση του κόστους και επιστρέφει το πρόγραμμα σε προηγούμενη επανάληψη εκτέλεσης εάν βρει επαναλήψεις λύσεων ίσες με τον αριθμό που του δίνεται. Για απενεργοποίηση παίρνει τιμή 0 (προκαθορισμένη τιμή), με τις επιλογές τιμών n > 0 να δίνουν τον αριθμό επαναλήψεων πριν από οπισθοδρόμηση.

repeatPerIteration: Χρησιμοποιείται για να αλλάξει την τιμή των επαναλήψεων που κρατά η παράμετρος επαναλήψεων repeat σε 0 ανάλογα με την τιμή που παίρνει. Για τιμή 0 (προκαθορισμένη τιμή) ο αριθμός των επαναλαμβανόμενων λύσεων που βρέθηκαν διατηρείται μεταξύ επαναλήψεων εκτέλεσης, ενώ με τιμή 1 ο αριθμός γίνεται 0 για ανίχνευση επαναλήψεων λύσεων μόνο στην τρέχον επανάληψη.

torchlightFails: Ενεργοποιεί τη χρήση του συστήματος TorchLight, μετρά πόσες φορές το σύστημα TorchLight δίνει προσέγγιση 0% και επιστρέφει σε προηγούμενη επανάληψη εκτέλεσης εάν μετρήσει απαντήσεις ίσες με τον αριθμό που του δίνεται. Για απενεργοποίηση παίρνει τιμή 0 (προκαθορισμένη τιμή), με τις επιλογές τιμών n > 0 να δίνουν τον αριθμό των φορών που μπορεί να μετρήσει απαντήσεις 0% πριν από οπισθοδρόμηση.

torchlightS: Ορίζει την παράμετρο -s στις εκτελέσεις του συστήματος TorchLight, η οποία ρυθμίζει τον αριθμό των καταστάσεων επιλογής που θα δοκιμαστούν από το σύστημα. Η προκαθορισμένη τιμή είναι 1000.

torchlightD: Ορίζει την παράμετρο -d στις εκτελέσεις του συστήματος TorchLight, η οποία ρυθμίζει το βάθος των καταστάσεων επιλογής που παίρνει το σύστημα για ανάλυση. Η προκαθορισμένη τιμή είναι 15.

backtrackAggression: Ορίζει τη στρατηγική που θα επιλεχτεί όσο αφορά τις οπισθοδρομήσεις που προκαλούνται από την παράμετρο επανάληψης (repeat) και αποτυχίας στην αναζήτηση του TorchLight (torchlightFails). Με τιμή 0 (προκαθορισμένη τιμή) η εκτέλεση μετά από οπισθοδρόμηση επιστρέφει στην προηγούμενη επανάληψη μείον ενός μετρητή που αυξάνεται για κάθε οπισθοδρόμηση. Για τιμή 1 επιλέγεται μια πιο δραστική στρατηγική, η οποία επιστρέφει την εκτέλεση στην επανάληψη i/2 μείον του προηγούμενου μετρητή. Για τιμή 2 επιλέγεται η πιο δραστική στρατηγική, κατά την οποία η εκτέλεση επιστρέφει στην επανάληψη 1 και κάθε 2 φορές που επιστρέφει στην επανάληψη 1 η τιμή της παραμέτρου χαλαρότητας k περιορίζεται κατά 1 (εκτός αν είναι 2).

backtrackTolerance: Χρησιμοποιείται για να αυξήσει τον αριθμό των επιτρεπόμενων επαναλήψεων λύσεων της παραμέτρου επανάληψης (repeat) ή τον αριθμό των αποτυχιών στην αναζήτηση του TorchLight (torchlightFails), όταν γίνεται οπισθοδρόμηση στην επανάληψη 1. Για τιμή 0 (προκαθορισμένη τιμή) δεν γίνεται καμία αύξηση, άρα οι παράμετροι παραμένουν με τις τιμές που τους έχουμε ορίσει αρχικά. Για επιλογές τιμών n > 0 αναλόγως ποιας παραμέτρου προκάλεσε την οπισθοδρόμηση, αυξάνουμε την τρέχων τιμή αυτής της παραμέτρου κατά την τιμή που ορίζουμε.

Κεφάλαιο 5

Πειραματική Μελέτη Παραμέτρων

5.1 Πειραματική Διαδικασία

Για να δοκιμάσουμε τις νέες παραμέτρους που υλοποιήθηκαν στο σύστημα χρειάζεται πειραματική μελέτη σε διάφορα προβλήματα. Σκοπός της πειραματικής μελέτης είναι να δούμε αν οι νέες παράμετροι επιφέρουν κάποιες βελτιώσεις στην αναζήτηση για λύση.

Για την πειραματική αξιολόγηση θα χρησιμοποιηθούν κλασικά προβλήματα Προγραμματισμού Δράσης όπως τα Trucks, Storage, Satellite και άλλα. Οι πλείστες παράμετροι που υλοποιήθηκαν σχετίζονται με αδιέξοδα τα οποία βρίσκει ο επιλυτής κατά τη διάρκεια της αναζήτησης. Έτσι, για την πειραματική μελέτη επιλέχτηκαν τα προβλήματα Trucks ως κύρια προβλήματα, λόγω του μεγάλου αριθμού αδιεξόδων που υπάρχουν στην εκτέλεση τους.

Για κάθε πρόβλημα θα γίνει πρώτα εκτέλεση χωρίς τις νέες υλοποιημένες παραμέτρους για σύγκριση αποτελεσμάτων. Έπειτα θα δοκιμαστεί η εκτέλεση με χρήση των νέων παραμέτρων που αφορούν την παράμετρο χαλαρότητας. Επίσης θα δοκιμαστεί η

επίδοση των παραμέτρων που σχετίζονται με την νέα παράμετρο επανάληψης, καθώς και η επίδοση των παραμέτρων που αφορούν την νέα παράμετρο ανάλυσης του συστήματος TorchLight. Μετά τις δοκιμές αυτές θα γίνουν επιπλέον δοκιμές στα προβλήματα με συνδυασμό παραμέτρων. Ο συνδυασμός αυτός θα δοκιμαστεί και σε άλλα προβλήματα εκτός των Trucks.

Για επιπλέον ανάλυση των παραμέτρων που σχετίζονται με οπισθοδρομήσεις θα χρησιμοποιηθεί το σύστημα Προγραμματισμού Δράσης Fast Downward. Κάθε φορά που θα γίνει οπισθοδρόμηση θα καλείται το σύστημα, το οποίο θα προσπαθήσει να βρει λύση στο πρόβλημα στην τρέχον επανάληψη με όριο χρόνου 300 δευτερόλεπτα. Εάν το σύστημα αυτό δεν καταφέρει να βρει λύση, τότε μετράμε την οπισθοδρόμηση ως σωστή, διαφορετικά ως λάθος, αφού σημαίνει ότι μπορούσαμε να προχωρήσουμε από τη συγκεκριμένη επανάληψη χωρίς οπισθοδρόμηση. Με αυτό τον τρόπο θα πάρουμε επιπλέον στατιστικά όσο αφορά το ποσοστό των οπισθοδρομήσεων που ήταν απαραίτητες.

Στις εκτελέσεις θα χρησιμοποιηθεί μόνο ευρετικό κόστους, λόγω του περιορισμένου χρόνου για πειράματα και του μεγάλου χρόνου που χρειάζονται τα προβλήματα να εκτελεστούν.

5.2 Πειραματικά Αποτελέσματα

Όπως προαναφέρθηκε, αρχικά θα δούμε την επίδοση του συστήματος στα προβλήματα Trucks χωρίς τις νέες παραμέτρους που υλοποιήσαμε. Θα γίνει χρήση της παραμέτρου επανεκκίνησης, για να δούμε πόσα προβλήματα επιλύνονται μόνο με χρήση της, και στη συνέχεια θα χρησιμοποιήσουμε τις νέες παραμέτρους που υλοποιήθηκαν. Η παράμετρος επανεκκίνησης μπορεί να επιφέρει λύση στις περιπτώσεις που έχουμε αδιέξοδο και το κόστος που επιστρέφει ο επιλυτής είναι 0 λόγω αποτυχίας ικανοποίησης των περιορισμών. Δεν χρειάζεται να γίνει εκτέλεση χωρίς αυτή την παράμετρο, αφού στις περιπτώσεις που έχουμε αδιέξοδα θα έχουμε αποτυχία κάθε φορά χωρίς αυτή, άρα μόνο να συνεισφέρει μπορεί στα αποτελέσματα.

Στα πειράματα γίνεται επανεκκίνηση κάθε 3 φορές που έχουμε αποτυχία, δηλαδή χρησιμοποιούμε reset = 3. Επίσης χρησιμοποιήθηκε όριο εκτέλεσης 600 δευτερολέπτων, αφού η παράμετρος επανεκκίνησης μπορεί να ξεκινά ξανά το πρόγραμμα από την αρχή συνέχεια αν βρίσκει συνέχεια αδιέξοδα μετά από οπισθοδρομήσεις. Τα αποτελέσματα της πειραματικής μελέτης της παραμέτρου επανεκκίνησης φαίνονται στον Πίνακα 5.1.

Πρόβλημα	Επιτυχία/Αποτυχία	Λόγος Αποτυχίας	Χρόνος Εκτέλεσης (δευτερόλεπτα)
p01	Επιτυχία	-	2
p02	Επιτυχία	-	6
p03	Επιτυχία	-	12
p04	Επιτυχία	-	14
p05	Επιτυχία	-	16
p06	Επιτυχία	-	41
p07	Αποτυχία	Αδιέξοδο	-
p08	Επιτυχία	-	43
p09	Αποτυχία	Αδιέξοδο	-
p10	Αποτυχία	Αδιέξοδο	-
p11	Αποτυχία	Αδιέξοδο	-
p12	Αποτυχία	Αδιέξοδο	-
p13	Αποτυχία	Αδιέξοδο	-
p14	Αποτυχία	Αδιέξοδο	-
p15	Αποτυχία	Αδιέξοδο	-
p16	Αποτυχία	Αδιέξοδο	-
p17	Αποτυχία	Αδιέξοδο	-
p18	Αποτυχία	Τέλος Χρόνου	-
p19	Αποτυχία	Αδιέξοδο	-
p20	Αποτυχία	Τέλος Χρόνου	-

Πίνακας 5.1 Αποτελέσματα παραμέτρου επανεκκίνησης στα Trucks

Όπως παρατηρούμε, στα πλείστα προβλήματα η αναζήτηση φτάνει σε αδιέξοδο με αποτέλεσμα η εκτέλεση να μην μπορεί να συνεχίσει και τελικά να αποτυγχάνει λόγω του ορίου αυστηρών βημάτων (30 βήματα με αυξήσεις των 2 κάθε φορά) στο οποίο φτάνει. Σε κάποιες περιπτώσεις έχουμε τέλος χρόνου, αφού το πρόγραμμα ξαναξεκινά από την επανάληψη 1 συνεχόμενα, λόγω της παραμέτρου επανεκκίνησης που βρίσκει αδιέξοδο ή λόγω του χρόνου που χρειάζεται το πρόβλημα. Το ποσοστό επιτυχίας επίλυσης των προβλημάτων ήταν 35%.

Αφού έχουμε τα αποτελέσματα των εκτελέσεων με την παράμετρο επανεκκίνησης, θα την απενεργοποιήσουμε για να δοκιμάσουμε τη συνεισφορά των υπόλοιπων παραμέτρων.

5.2.1 Εκτελέσεις με Παράμετρο Χαλαρότητας Καταστάσεων

Για να δούμε τη συνεισφορά της μεταβολής της παραμέτρου χαλαρότητας στην εκτέλεση θα χρησιμοποιήσουμε μια αρχικά μεγάλη τιμή για την παράμετρο χαλαρότητας ίση με 7, με μεταβολή της κάθε 2 επαναλήψεις (μετρώντας και πιθανές οπισθοδρομήσεις). Δηλαδή θα χρησιμοποιήσουμε k = 7, kReduction = 2 και kReductionDirection = 1. Το χρονικό όριο για την εκτέλεση θα είναι πάλι 600 δευτερόλεπτα για τις περιπτώσεις που η επίλυση του προβλήματος μπορεί να είναι χρονοβόρα. Τα αποτελέσματα της πειραματικής μελέτης της παραμέτρου χαλαρότητας καταστάσεων φαίνονται στον Πίνακα 5.2.

Πρόβλημα	Επιτυχία/Αποτυχία	Λόγος Αποτυχίας	Χρόνος Εκτέλεσης (δευτερόλεπτα)
p01	Επιτυχία	-	2
p02	Επιτυχία	-	6
p03	Επιτυχία	-	11
p04	Επιτυχία	-	16
p05	Επιτυχία	-	16
p06	Αποτυχία	Αδιέξοδο	-
p07	Αποτυχία	Αδιέξοδο	-

p08	Επιτυχία	-	38
p09	Αποτυχία	Αδιέξοδο	-
p10	Αποτυχία	Αδιέξοδο	-
p11	Αποτυχία	Αδιέξοδο	-
p12	Αποτυχία	Αδιέξοδο	-
p13	Αποτυχία	Αδιέξοδο	-
p14	Αποτυχία	Αδιέξοδο	-
p15	Αποτυχία	Αδιέξοδο	-
p16	Αποτυχία	Αδιέξοδο	-
p17	Αποτυχία	Αδιέξοδο	-
p18	Αποτυχία	Αδιέξοδο	-
p19	Αποτυχία	Αδιέξοδο	-
p20	Αποτυχία	Τέλος Χρόνου	-

Πίνακας 5.2 Αποτελέσματα παραμέτρου χαλαρότητας καταστάσεων στα Trucks

Όπως φαίνεται τα αποτελέσματα εκτέλεσης είναι παρόμοια με τα προηγούμενα, με κάποιες διαφορές στις αποτυχίες, οι οποίες μπορεί να οφείλονται στο μονοπάτι που επιλέχτηκε από τον επιλυτή για την αναζήτηση. Επίσης σε μια περίπτωση που είχαμε τέλος χρόνου εδώ έχουμε αποτυχία, αφού δεν χρησιμοποιείται οποιαδήποτε παράμετρος οπισθοδρόμησης. Το ποσοστό επιτυχίας επίλυσης των προβλημάτων ήταν 30%. Η παράμετρος χαλαρότητας καταστάσεων εδώ δεν μας έδωσε κάποια συνεισφορά στην εκτέλεση. Η συνδυαστική χρήση της όμως με άλλες νέες παραμέτρους μπορεί να βοηθήσει. Αυτό θα δοκιμαστεί αργότερα, μετά την εξέταση των άλλων παραμέτρων μόνων τους.

Για τα επόμενα πειράματα για τις άλλες παραμέτρους η παράμετρος χαλαρότητας καταστάσεων θα παραμένει σταθερή στον αριθμό 4, για να μπορούν να δοκιμαστούν χωρίς μεταβολή της.

5.2.2 Εκτελέσεις με Παράμετρο Επανάληψης

Για την παράμετρο επανάληψης θα γίνουν διαφορετικοί συνδυασμοί των παραμέτρων που σχετίζονται μαζί της. Η πρώτη δοκιμή είναι με επιλογή στρατηγικής για οπισθοδρόμηση μικρής δραστικότητας, δηλαδή με οπισθοδρόμηση στην προηγούμενη επανάληψη (μείον του μετρητή). Ο αριθμός των επιτρεπόμενων επαναλήψεων λύσεων που θα δηλωθεί με την παράμετρο επαναλήψεων είναι 3. Κάθε φορά που γίνεται οπισθοδρόμηση στην πρώτη επανάληψη θα αυξάνουμε τον επιτρεπόμενο αριθμό επαναλήψεων λύσεων κατά 1. Ο αριθμός των επαναλήψεων που βρέθηκαν θα διατηρείται μεταξύ επαναλήψεων. Δηλαδή θα χρησιμοποιήσουμε τις παραμέτρους backtrackAggression = 0, repeat = 3, backtrackTolerance = 1 και repeatPerIteration = 0.

Υπάρχουν περιπτώσεις που μπορεί να χρειαστεί πολύς χρόνος για να βρεθεί μια λύση, αφού με την παράμετρο επανάληψης θα ανιχνεύονται πιθανά αδιέξοδα και θα γίνεται συνεχώς οπισθοδρόμηση μέχρι να βρεθεί λύση ή να οδηγηθούμε στο όριο των αυστηρών βημάτων μετά από πολλές προσπάθειες. Για αποφυγή αυτού του φαινομένου θα θέσουμε όριο εκτέλεσης ξανά στα 600 δευτερόλεπτα και στις περιπτώσεις που φτάνουμε το επιτρεπόμενο όριο θα θεωρούμε την εκτέλεση ως αποτυχία.

Επίσης όπως προαναφέρθηκε θα χρησιμοποιούμε το σύστημα Fast Downward για να δούμε το ποσοστό των οπισθοδρομήσεων (εάν υπήρχαν) που ήταν σωστές, όπως αυτό ορίστηκε προηγουμένως. Τα αποτελέσματα της πρώτης πειραματικής μελέτης της παραμέτρου επανάληψης με τις υπόλοιπες παραμέτρους που αναφέρθηκαν φαίνονται στον Πίνακα 5.3.

Πρόβλημα	Επιτυχία/Αποτυχία	Λόγος Αποτυχίας	Χρόνος Εκτέλεσης (δευτερόλεπτα)	Ποσοστό Σωστών Οπισθοδρομήσεων
p01	Επιτυχία	-	3	-
p02	Επιτυχία	-	6	-
p03	Επιτυχία	-	12	-
p04	Επιτυχία	-	15	-

p05	Επιτυχία	-	18	-
p06	Επιτυχία	-	55	100%
p07	Αποτυχία	Τέλος Χρόνου	-	90.9%
p08	Επιτυχία	-	34	100%
p09	Επιτυχία	-	394	100%
p10	Αποτυχία	Τέλος Χρόνου	-	86.6%
p11	Επιτυχία	-	512	87.5%
p12	Αποτυχία	Τέλος Χρόνου	-	93.7%
p13	Αποτυχία	Τέλος Χρόνου	-	88.2%
p14	Αποτυχία	Τέλος Χρόνου	-	92.3%
p15	Επιτυχία	-	503	86.6%
p16	Αποτυχία	Τέλος Χρόνου	-	100%
p17	Αποτυχία	Τέλος Χρόνου	-	100%
p18	Αποτυχία	Τέλος Χρόνου	-	100%
p19	Αποτυχία	Τέλος Χρόνου	-	100%
p20	Αποτυχία	Τέλος Χρόνου	-	-

Πίνακας 5.3 Αποτελέσματα πρώτης δοκιμής παραμέτρων επανάληψης στα Trucks

Όπως φαίνεται, ορισμένα προβλήματα που δεν μπορούσαν να λυθούν λόγω των αδιεξόδων κατάφεραν να λυθούν με χρήση των παραμέτρων επανάληψης. Το ποσοστό επιτυχίας επίλυσης των προβλημάτων στην πρώτη δοκιμή παραμέτρων επανάληψης ήταν 50%. Επίσης μπούμε να δούμε ότι γενικά το ποσοστό των σωστών οπισθοδρομήσεων που εκτελούνται είναι ψηλό. Σε αρκετά προβλήματα φτάσαμε σε τέλος χρόνου, άρα θεωρούνται ως αποτυχίες, αλλά παρόλο αυτού οι σωστές οπισθοδρομήσεις δεν ήταν χαμηλές.

Η δεύτερη δοκιμή των παραμέτρων επανάληψης θα γίνει με επιλογή στρατηγικής για οπισθοδρόμηση μεγάλης δραστικότητας, δηλαδή με οπισθοδρόμηση στην πρώτη επανάληψη και περιορισμό της παραμέτρου χαλαρότητας καταστάσεων (που ξεκινά με 4) κάθε 2 φορές που φτάνουμε στην πρώτη επανάληψη. Ο αριθμός των επιτρεπόμενων επαναλήψεων λύσεων που θα δηλωθεί με την παράμετρο επαναλήψεων είναι 4, έτσι ώστε να επιτρέπουμε παραπάνω προσπάθειες στον επιλυτή πριν δραστικά

επιστρέψουμε στην πρώτη επανάληψη. Κάθε φορά που γίνεται οπισθοδρόμηση στην πρώτη επανάληψη θα αυξάνουμε τον επιτρεπόμενο αριθμό επαναλήψεων λύσεων κατά 1. Ο αριθμός των επαναλήψεων που βρέθηκαν θα διατηρείται μεταξύ επαναλήψεων. Δηλαδή θα χρησιμοποιήσουμε τις παραμέτρους backtrackAggression = 2, repeat = 5, backtrackTolerance = 1 και repeatPerIteration = 0. Τα αποτελέσματα της δεύτερης πειραματικής μελέτης της παραμέτρου επανάληψης φαίνονται στον Πίνακα 5.4.

Πρόβλημα	Επιτυχία/Αποτυχία	Λόγος Αποτυχίας	Χρόνος Εκτέλεσης (δευτερόλεπτα)	Ποσοστό Σωστών Οπισθοδρομήσεων
p01	Επιτυχία	-	2	-
p02	Επιτυχία	-	5	-
p03	Επιτυχία	-	12	-
p04	Επιτυχία	-	13	-
p05	Επιτυχία	-	16	-
p06	Επιτυχία	-	41	100%
p07	Επιτυχία	-	404	93.7%
p08	Επιτυχία	-	40	100%
p09	Επιτυχία	-	288	94.4%
p10	Αποτυχία	Τέλος Χρόνου	-	94.1%
p11	Επιτυχία	-	491	88.8%
p12	Αποτυχία	Τέλος Χρόνου	-	94.7%
p13	Επιτυχία	-	549	90.4%
p14	Αποτυχία	Τέλος Χρόνου	-	95%
p15	Αποτυχία	Τέλος Χρόνου	-	91.6%
p16	Αποτυχία	Τέλος Χρόνου	-	92.3%
p17	Αποτυχία	Τέλος Χρόνου	-	100%
p18	Αποτυχία	Τέλος Χρόνου	-	100%
p19	Αποτυχία	Τέλος Χρόνου	-	100%
p20	Αποτυχία	Τέλος Χρόνου	-	-

Πίνακας 5.4 Αποτελέσματα δεύτερης δοκιμής παραμέτρων επανάληψης στα Trucks

Στη δεύτερη δοκιμή λύθηκαν διαφορετικά προβλήματα από την πρώτη δοκιμή. Για παράδειγμα, το πρόβλημα 7 δεν λύθηκε στην πρώτη δοκιμή, αλλά λύθηκε στη δεύτερη. Αντιθέτως, προβλήματα όπως το 15 λύθηκαν στην πρώτη δοκιμή, αλλά δεν λύθηκαν στη δεύτερη. Το ποσοστό επιτυχίας επίλυσης των προβλημάτων στη δεύτερη δοκιμή παραμέτρων επανάληψης ήταν 55%. Το ποσοστό των σωστών οπισθοδρομήσεων παραμένει ψηλό.

Η τρίτη δοκιμή των παραμέτρων επανάληψης θα γίνει με επιλογή στρατηγικής για οπισθοδρόμηση μεσαίας δραστικότητας, δηλαδή με οπισθοδρόμηση στην επανάληψη i/2 (μείον του μετρητή). Ο αριθμός των επιτρεπόμενων επαναλήψεων λύσεων που θα δηλωθεί με την παράμετρο επαναλήψεων είναι 3. Κάθε φορά που γίνεται οπισθοδρόμηση στην πρώτη επανάληψη θα αυξάνουμε τον επιτρεπόμενο αριθμό επαναλήψεων λύσεων κατά 1. Ο αριθμός των επαναλήψεων που βρέθηκαν θα διατηρείται μεταξύ επαναλήψεων. Δηλαδή θα χρησιμοποιήσουμε τις παραμέτρους backtrackAggression = 1, repeat = 4, backtrackTolerance = 1 και repeatPerIteration = 0. Τα αποτελέσματα της τρίτης πειραματικής μελέτης της παραμέτρου επανάληψης φαίνονται στον Πίνακα 5.5.

Πρόβλημα	Επιτυχία/Αποτυχία	Λόγος Αποτυχίας	Χρόνος Εκτέλεσης (δευτερόλεπτα)	Ποσοστό Σωστών Οπισθοδρομήσεων
p01	Επιτυχία	-	2	-
p02	Επιτυχία	-	6	-
p03	Επιτυχία	-	13	-
p04	Επιτυχία	-	18	-
p05	Επιτυχία	-	16	-
p06	Επιτυχία	-	37	-
p07	Επιτυχία	-	309	90.9%
p08	Επιτυχία	-	53	100%
p09	Επιτυχία	-	225	100%
p10	Αποτυχία	Τέλος Χρόνου	-	88.2%
p11	Επιτυχία	-	412	94.4%

p12	Αποτυχία	Τέλος Χρόνου	-	95%
p13	Αποτυχία	Τέλος Χρόνου	-	94.1%
p14	Αποτυχία	Τέλος Χρόνου	-	88.8%
p15	Επιτυχία	-	470	92.8%
p16	Αποτυχία	Τέλος Χρόνου	-	85.7%
p17	Αποτυχία	Τέλος Χρόνου	-	100%
p18	Αποτυχία	Τέλος Χρόνου	-	100%
p19	Αποτυχία	Τέλος Χρόνου	-	100%
p20	Αποτυχία	Τέλος Χρόνου	-	-

Πίνακας 5.5 Αποτελέσματα τρίτης δοκιμής παραμέτρων επανάληψης στα Trucks

Στην τρίτη δοκιμή πάλι λύθηκαν διαφορετικά προβλήματα από τις προηγούμενες δοκιμές. Το ποσοστό επιτυχίας επίλυσης των προβλημάτων στην τρίτη δοκιμή παραμέτρων επανάληψης ήταν 55%. Το ποσοστό των σωστών οπισθοδρομήσεων και σε αυτή τη δοκιμή ήταν ψηλό. Επίσης, μπορούμε να δούμε ότι σε προβλήματα που λύθηκαν σε προηγούμενες δοκιμές τώρα έχουμε μικρότερο χρόνο επίλυσης.

Συγκρίνοντας τα αποτελέσματα από τις δοκιμές παραμέτρων επανάληψης, μπορούμε να δούμε ότι την καλύτερη επίδοση είχαν οι πιο δραστικές στρατηγικές σε αυτά τα προβλήματα. Ο αριθμός των προβλημάτων που κατάφεραν να λυθούν διαφέρει κατά 1 μεταξύ των στρατηγικών και ήταν μεγαλύτερος και στις τρεις δοκιμές συγκριτικά με την υλοποίηση της παραμέτρου επανεκκίνησης.

Αφού τελειώσαμε με τις παραμέτρους επανάληψης, θα τις απενεργοποιήσουμε για επιπλέον δοκιμές με άλλες παραμέτρους.

5.2.3 Εκτελέσεις με Παράμετρο Ανάλυσης TorchLight

Για δοκιμή της παραμέτρου ανάλυσης αποτυχιών στην αναζήτηση του TorchLight θα χρησιμοποιήσουμε επιτρεπτή τιμή προσεγγίσεων 0% ίση με 2 και τιμές για τις παραμέτρους -s και -d του TorchLight 1000 και 15 αντίστοιχα. Η στρατηγική για οπισθοδρόμηση που θα χρησιμοποιηθεί είναι αυτή με μικρή δραστικότητα, στην οποία

γίνεται οπισθοδρόμηση στην προηγούμενη επανάληψη (μείον του μετρητή). Δεν θα αυξάνουμε τον επιτρεπόμενο αριθμό αποτυχιών όταν οπισθοδρομούμε στην πρώτη επανάληψη. Δηλαδή θα χρησιμοποιήσουμε τις παραμέτρους torchlightFaçls = 2, torchlightS = 1000, torchlightD = 15, backtrackAggression = 0 και backtrackTolerance = 0.

Το χρονικό όριο της εκτέλεσης είναι πάλι 600 δευτερόλεπτα και θα γίνει ξανά χρήση του συστήματος Fast Downward για μέτρηση των σωστών οπισθοδρομήσεων. Τα αποτελέσματα της πειραματικής μελέτης της παραμέτρου ανάλυσης TorchLight με τις υπόλοιπες παραμέτρους που αναφέρθηκαν φαίνονται στον Πίνακα 5.6.

Πρόβλημα	Επιτυχία/Αποτυχία	Λόγος Αποτυχίας	Χρόνος Εκτέλεσης (δευτερόλεπτα)	Ποσοστό Σωστών Οπισθοδρομήσεων
p01	Επιτυχία	-	3	-
p02	Επιτυχία	-	5	-
p03	Επιτυχία	-	12	-
p04	Επιτυχία	-	14	-
p05	Επιτυχία	-	16	-
p06	Επιτυχία	-	44	100%
p07	Αποτυχία	Αδιέξοδο	-	88.8%
p08	Επιτυχία	-	39	-
p09	Επιτυχία	-	336	90.4%
p10	Αποτυχία	Τέλος Χρόνου	-	94.1%
p11	Αποτυχία	Τέλος Χρόνου	-	83.3%
p12	Αποτυχία	Τέλος Χρόνου	-	82.3%
p13	Αποτυχία	Τέλος Χρόνου	-	84.2%
p14	Αποτυχία	Τέλος Χρόνου	-	90.9%
p15	Αποτυχία	Τέλος Χρόνου	-	85.7%
p16	Αποτυχία	Τέλος Χρόνου	-	90%
p17	Αποτυχία	Αδιέξοδο	-	-
p18	Αποτυχία	Αδιέξοδο	-	-

p19	Αποτυχία	Αδιέξοδο	-	-
p20	Αποτυχία	Τέλος Χρόνου	-	-

Πίνακας 5.6 Αποτελέσματα Δοκιμής Παραμέτρων Ανάλυσης TorchLight σε Προβλήματα Trucks

Με τις παραμέτρους που σχετίζονται με την ανάλυση του TorchLight καταφέραμε να πάρουμε ποσοστό επιτυχίας επίλυσης των προβλημάτων 45%. Το αποτέλεσμα αυτό συγκριτικά με τις παραμέτρους επανάληψης είναι πιο χαμηλό. Επίσης, μπούμε να δούμε ότι γενικά το ποσοστό των σωστών οπισθοδρομήσεων που γίνονται είναι και αυτό συγκριτικά πιο χαμηλό από προηγουμένως, αλλά παραμένει ψηλό. Εδώ παρατηρούμε ακόμα ότι το σύστημα TorchLight δεν εντοπίζει όλα τα αδιέξοδα με την προσέγγιση του, αφού σε κάποιες περιπτώσεις έχουμε αποτυχία από αδιέξοδο. Σε αυτές τις περιπτώσεις δεν αναγνωρίστηκε το αδιέξοδο για οπισθοδρόμηση.

5.2.4 Εκτελέσεις με Συνδυασμούς Παραμέτρων

Αφού δοκιμάσαμε ομάδες παραμέτρων που σχετίζονται μεταξύ τους σε πειράματα, τώρα μπορούμε να χρησιμοποιήσουμε ένα συνδυασμό όλων των νέων υλοποιημένων παραμέτρων μαζί για να δούμε τα αποτελέσματα.

Όσον αφορά την παράμετρο χαλαρότητας στην εκτέλεση, θα χρησιμοποιήσουμε μια αρχικά μεγάλη τιμή 7 με μεταβολή της κάθε 2 επαναλήψεις προς μεγαλύτερους αριθμούς επαναλήψεων (δηλαδή δεν μετράμε επανάληψη όταν οπισθοδρομήσουμε). Άρα, θα χρησιμοποιήσουμε k = 7, kReduction = 2 και kReductionDirection = 0.

Για την παράμετρο επανάληψης θα χρησιμοποιήσουμε συνδυαστικές τιμές από τις προηγούμενες δοκιμές. Η τιμή των επιτρεπόμενων επαναλήψεων λύσεων θα είναι 3 με μεσαία στρατηγική οπισθοδρόμησης για επιστροφή στην επανάληψη i/2 (μείον του μετρητή). Κάθε φορά που γίνεται οπισθοδρόμηση στην πρώτη επανάληψη θα αυξάνουμε τον επιτρεπόμενο αριθμό επαναλήψεων λύσεων κατά 1. Ο αριθμός των επαναλήψεων που βρέθηκαν θα διατηρείται μεταξύ επαναλήψεων. Δηλαδή θα

χρησιμοποιήσουμε τις παραμέτρους repeat = 3, backtrackAggression = 1, backtrackTolerance = 1 και repeatPerIteration = 0.

Για την παράμετρο ανάλυσης αποτυχιών στην αναζήτηση του TorchLight θα χρησιμοποιήσουμε επιτρεπτή τιμή προσεγγίσεων 0% ίση με 2 και τιμές για τις παραμέτρους -s και -d ίσες με 1000 και 15 αντίστοιχα. Η στρατηγική οπισθοδρόμησης για την παράμετρο ανάλυσης του TorchLight θα είναι η ίδια με την παράμετρο επανάληψης, αφού θα είναι ενεργοποιημένες μαζί.

Όπως είδαμε προηγουμένως με το Fast Downward, οι παράμετροι γενικά δίνουν αρκετά καλά αποτελέσματα στις σωστές οπισθοδρομήσεις σε αυτό το πρόβλημα. Άρα, στα ακόλουθα πειράματα δεν θα χρησιμοποιήσουμε το σύστημα Fast Downward και θα χρησιμοποιήσουμε τον επιπλέον χρόνο για να αυξήσουμε τον επιτρεπτό χρόνο επίλυσης των προβλημάτων σε 1800 δευτερόλεπτα. Τα αποτελέσματα της πειραματικής μελέτης του συνδυασμού των παραμέτρων που επιλέξαμε φαίνονται στον Πίνακα 5.7.

Πρόβλημα	Επιτυχία/Αποτυχία	Λόγος Αποτυχίας	Χρόνος Εκτέλεσης (δευτερόλεπτα)
p01	Επιτυχία	-	2
p02	Επιτυχία	-	4
p03	Επιτυχία	-	13
p04	Επιτυχία	-	15
p05	Επιτυχία	-	18
p06	Επιτυχία	-	-
p07	Επιτυχία	-	399
p08	Επιτυχία	-	38
p09	Επιτυχία	-	262
p10	Αποτυχία	Αδιέξοδο	-
p11	Επιτυχία	-	521
p12	Αποτυχία	Αδιέξοδο	-
p13	Επιτυχία	-	723
p14	Επιτυχία	-	1542

p15	Επιτυχία	-	495
p16	Αποτυχία	Τέλος Χρόνου	-
p17	Αποτυχία	Τέλος Χρόνου	-
p18	Αποτυχία	Τέλος Χρόνου	-
p19	Αποτυχία	Τέλος Χρόνου	-
p20	Αποτυχία	Τέλος Χρόνου	-

Πίνακας 5.7 Αποτελέσματα Συνδυασμού Παραμέτρων σε Προβλήματα Trucks

Παρατηρούμε ότι με το μεγαλύτερο όριο χρόνου και τον συνδυασμό τον παραμέτρων κατάφεραν να λυθούν 65% των προβλημάτων. Το πρόβλημα 14 δεν είχε καταφέρει να επιλυθεί σε κανένα προηγούμενο πείραμα, αλλά λύθηκε με τον επιπλέον χρόνο που δώσαμε εδώ. Αξίζει να σημειωθεί όμως ότι με τον προηγούμενο επιτρεπτό χρόνο επίλυσης των 600 δευτερολέπτων θα είχαμε λύση 55% των προβλημάτων που ήταν το μέγιστο ποσοστό που είχαμε πάρει από τα προηγούμενα πειράματα.

Μια άλλη παρατήρηση είναι ότι έχουμε αποτυχίες λόγω αδιεξόδων με τον μεγαλύτερο χρόνο επίλυσης. Αυτό συμβαίνει αφού κάθε φορά που οπισθοδρομούμε στην επανάληψη 1 οι επιτρεπτές επαναλήψεις λύσεων ή 0% προσεγγίσεις του TorchLight ανάλογα αυξάνονται κατά 1. Σε μεγάλο χρονικό διάστημα, εάν έχουμε πολλές οπισθοδρομήσεις στην επανάληψη 1, τότε θα έχουν πλέον μεγάλο αριθμό που δεν θα είναι αρκετά περιοριστικός για να οπισθοδρομεί την εκτέλεση όταν πρέπει. Άρα, το πρόγραμμα θα μείνει σε αδιέξοδο μέχρι να φτάσουμε τον μέγιστο αριθμό αυστηρών βημάτων που θα τερματίσει την εκτέλεση.

Θα κρατήσουμε αυτό τον συνδυασμό τιμών των παραμέτρων για να δοκιμάσουμε την επίδοση τους και σε άλλα προβλήματα εκτός από τα Trucks. Τα προβλήματα που θα χρησιμοποιηθούν είναι τα Storage, Satellite, Pipesworld και Rovers. Τα προβλήματα αυτά δεν έχουν πολλά αδιέξοδα όπως τα προβλήματα Trucks, όμως η ιδέα της οπισθοδρόμησης και της επανάληψης που παρατηρούν οι παράμετροι που υλοποιήσαμε μπορούν να βοηθήσουν στην επιτάχυνση της λύσης των προβλημάτων με απόρριψη συγκεκριμένων λύσεων.

Ο επιτρεπτός χρόνος εκτέλεσης για τα προβλήματα είναι 600 δευτερόλεπτα λόγω του μεγάλου αριθμού προβλημάτων. Οι αναζητήσεις που ξεπερνούν αυτό το όριο θα θεωρούνται ως αποτυχίες. Για σύγκριση θα χρησιμοποιήσουμε αποτελέσματα των προβλημάτων χωρίς χρήση οποιασδήποτε από τις νέες υλοποιημένες παραμέτρους. Τα αποτελέσματα των πειραμάτων φαίνονται στον Πίνακα 5.8 μαζί με τα προηγούμενα αποτελέσματα των Trucks για εκτελέσεις μικρότερες των 600 δευτερολέπτων.

Προβλήματα	Αριθμός Προβλημάτων	Ποσοστό Επιτυχίας χωρίς Νέες Παραμέτρους	Ποσοστό Επιτυχίας με Νέες Παραμέτρους
Storage	30	63.3%	66.6%
Satellite	36	36.1%	33.3%
Pipesworld	50	30%	32%
Rovers	40	75%	75%
Trucks	20	35%	55%

Πίνακας 5.8 Αποτελέσματα πειραμάτων σε διάφορα προβλήματα

Από τα πειράματα μπορούμε να δούμε ότι είχαμε μερική αύξηση του ποσοστού επιτυχίας με χρήση των παραμέτρων στα προβλήματα Storage και Pipesworld. Η αλλαγή όμως είναι μικρή συγκριτικά με την αύξηση του ποσοστού επιτυχίας που είχαμε στα προβλήματα Trucks που αναλύθηκαν προηγουμένως. Στα προβλήματα Rovers το ποσοστό επιτυχίας παραμένει το ίδιο, ενώ στα προβλήματα Satellite βλέπουμε μείωση στον αριθμό των προβλημάτων που κατάφεραν να λυθούν στο χρονικό πλαίσιο που δόθηκε. Αυτό μας δείχνει ότι αυτές οι νέες παράμετροι πρέπει να χρησιμοποιούνται κυρίως σε προβλήματα στα οποία υπάρχουν αδιέξοδα, αφού σε αυτά παρατηρήθηκε η μεγαλύτερη συνεισφορά τους. Οι οπισθοδρομήσεις σε προβλήματα που δεν έχουν αδιέξοδα μπορεί να μην είναι πάντα απαραίτητες.

Κεφάλαιο 6

Συμπεράσματα

6.1 Συμπεράσματα Διπλωματικής Εργασίας

Στο κεφάλαιο αυτό θα παρουσιάσουμε κάποια από τα συμπεράσματα που προέκυψαν από την πειραματική αξιολόγηση των επεκτάσεων που έγιναν στο σύστημα που χρησιμοποιήθηκε στα πλαίσια της εργασίας.

Όπως είχε αναφερθεί και στην εισαγωγή της εργασίας, σκοπός της ήταν να γίνουν επεκτάσεις διαφόρων ευρετικών μεθόδων του συστήματος που έχουν περιγραφτεί για βελτίωση των αποτελεσμάτων στην αναζήτηση λύσεων. Με τις νέες επεκτάσεις καταφέραμε να αντιμετωπίσουμε κάποια συγκεκριμένα προβλήματα που παρατηρήθηκαν κατά την εκτέλεση του συστήματος. Το κύριο πρόβλημα που προσπαθήσαμε να αντιμετωπίσουμε ήταν αυτό των αδιεξόδων στην εκτέλεση, όπου ο επιλυτής δεν μπορεί να προχωρήσει σε επόμενες επαναλήψεις, με αποτέλεσμα να μην μπορεί να βρει λύση στο πρόβλημα.

Οι τροποποιήσεις που έγιναν στο σύστημα, όπως φάνηκε και από τα πειράματα, μπορούν να συνεισφέρουν στην επίλυση αυτού του προβλήματος. Όπως είδαμε προηγουμένως, σε προβλήματα που έχουν πολλά αδιέξοδα η χρήση των νέων υλοποιημένων παραμέτρων, κυρίως συνδυαστικά μεταξύ τους, αυξάνει το ποσοστό επιτυχίας της επίλυσης. Οι παράμετροι αυτοί μπορούν να χρησιμοποιούνται και σε άλλα προβλήματα, αλλά δεν συνεισφέρουν τόσο όσο στα προβλήματα με αδιέξοδα.

6.2 Μελλοντική Εργασία

Κατά τη διάρκεια ανάπτυξης των νέων παραμέτρων του συστήματος υπήρχαν διάφορες ιδέες για μελλοντική επέκταση τους. Όπως είδαμε από τα πειράματα, ο συνδυασμός των διάφορων παραμέτρων συνεισφέρει περισσότερο σε κάποιες περιπτώσεις από την μη-συνδυαστική χρήση τους. Ως μελλοντική εργασία μπορεί να δοκιμαστούν περισσότεροι τρόποι συνδυασμών των παραμέτρων για διαφορετικά προβλήματα. Επίσης, μπορεί να γίνει χρήση χαλαρωμένων βημάτων ως ευρετικό.

Όπως είδαμε οι παράμετροι δεν λύνουν το πρόβλημα σε όλες τις περιπτώσεις. Κάποιες φορές το πρόβλημα δεν καταφέρνει να επιλυθεί σε εύλογο χρονικό διάστημα, με το ποσοστό των οπισθοδρομήσεων που είναι σωστές, παρόλο που είναι ψηλό, να μην είναι το μέγιστο δυνατό. Μπορεί να γίνει προέκταση αυτών των μεθόδων έτσι ώστε να έχουμε πιο ψηλό ποσοστό σωστών οπισθοδρομήσεων.

Αφού οι παράμετροι είναι ευρετικά, μπορούν να δημιουργηθούν νέοι παράμετροι που χρησιμοποιούν διαφορετικά δεδομένα όσο αφορά την αξιολόγηση μιας κατάστασης που βρισκόμαστε για επιλογή οπισθοδρόμησης. Έτσι και οι στρατηγικές που χρησιμοποιούνται μπορούν να επεκταθούν ώστε να έχουμε δυναμική επιλογή στρατηγικής ανάλογα με τις οπισθοδρομήσεις που έγιναν κατά τη διάρκεια της εκτέλεσης του συστήματος.

Βιβλιογραφία

[1] Y. Dimopoulos et al. “plasp 3: Towards Effective ASP Planning”, Theory and Practice of Logic Programming, 2019

[2] M. Gebser et al., “Answer Set Solving in Practice”, 2012

[3] M. Gebster et al., “A User’s Guide to gringo, clasp, clingo, and iclingo (version 3.x)”, 2010

[4] M. Gebster et al. “plasp: A Prototype for PDDL-Based Planning in ASP”, 2011

[5] M. Ghallab, D. Nau, and P. Traverso, “Automated Planning: Theory and Practice”, 1st Edition, Morgan Kaufmann, San Francisco, 2004.

[6] M. Ghallab et al., “PDDL - The Planning Domain Definition Language (Version 1.2)”, 1998

[7] J. Hoffmann, “The TorchLight Tool: Analyzing Search Topology Without Running Any Search”, INRIA, Nancy, France

[8] S. Russell and P. Norvig, “Artificial Intelligence: A Modern Approach”, 3rd Edition, Prentice Hall, New Jersey, 2010.

[9] S. Russell and P. Norvig, “Artificial Intelligence: A Modern Approach”, 2nd Edition, Prentice Hall, New Jersey, 2003.

[10] T. Schaub, University of Potsdam, All Lectures under Potassco Teaching https://potassco.org