Ατομική Διπλωματική Εργασία

ΜΕΘΟΔΟΙ ΒΕΛΤΙΩΣΗΣ ΠΟΙΟΤΗΤΑΣ ΠΛΑΝΩΝ (ΠΡΟΓΡΑΜΜΑΤΩΝ ΔΡΑΣΗΣ)

Φελλά Αικατερίνη

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάιος 2013

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μέθοδοι Βελτίωσης Ποιότητας Πλάνων (Προγραμμάτων Δράσης)

Φελλά Αικατερίνη

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 2013

Ευχαριστίες

Αρχικά, θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή της διπλωματικής μου εργασίας Δρ. Γιάννη Δημόπουλο για την βοήθεια, τις συμβουλές του και την καθοδήγηση που μου πρόσφερε στα πλάσια της διπλωματικής μου εργασίας.

Θα ήθελα να ευχαριστήσω την οικογένεια μου και ιδιαίτερα τους γονείς μου Νεόφυτο και Παναγιώτα, τις αδερφές μου Χριστοφόρα και Αργυρώ καθώς επίσης και τον Γιώργο για την αμέριστη συμπαράσταση και την στήριξη τους καθ' όλη τη διάρκεια των σπουδών μου.

Τέλος, θα ήθελα να ευχαριστήσω τους φίλους μου για την στήριξη και την βοήθεια που μου παρείχαν σε αυτό το σημαντικό κομμάτι της ζωής μου.

Περίληψη

Η παρούσα διπλωματική εργασία αποσκοπεί στην υλοποίηση μεθόδων μέσω διαφόρων εργαλείων έτσι ώστε να επιτευχθεί η μείωση του συνολικού αριθμού των ενεργειών που υπάρχουν σε ένα αρχείο ενεργειών. Μέσα από τη μελέτη διαφόρων άρθρων και την εξοικείωση με τα εργαλεία τα οποία χρησιμοποιήθηκαν έγινε αντιληπτή η έννοια των προβλημάτων Προγραμματισμού Δράσης.

Ο στόχος της διπλωματικής εργασίας έχει υλοποιηθεί σε ένα ενιαίο πρόγραμμα. Για την υλοποίηση του προγράμματος αυτού χρησιμοποιήθηκαν τα εργαλεία LAMA2011 το οποίο είχε σαν αποτέλεσμα ένα αρχείο ενεργειών καθώς επίσης και το εργαλείο SATPLAN2006 με κωδικοποιητή SMP (SAT-MAX-PLAN) το οποίο χρησιμοποιήθηκε για να δίνει ένα αρχείο ενεργειών στην έξοδο μετά από μία επεξεργασία του αρχείου προβλήματος.

Στο πρώτο μέρος υλοποίησης υλοποιήθηκε ένα πρόγραμμα στο οποίο γινόταν τροποποίηση του αρχείου προβλήματος. Στο πρόγραμμα αυτό γινόταν η δημιουργία μέσω διαφόρων μεθόδων μία λίστα διαγραφής καταστάσεων και μία λίστα προσθήκης για το αρχείο προβλήματος. Οι καταστάσεις που υπάρχουν στις δύο λίστες αφορούν τις καταστάσεις που υπάρχουν στο αρχείο προβλήματος. Οι καταστάσεις που υπήρχαν στην λίστα διαγραφής τότε αντικαθιστώνται από τις καταστάσεις που υπάρχουν στη λίστα προσθήκης. Στο δεύτερο μέρος υλοποίησης υλοποιήθηκε το πρόγραμμα στο οποίο γινόταν ο διαχωρισμός του αρχείου ενεργειών σε υποσύνολα ανάλογα με τις γραμμές του αρχείου. Στο τρίτο μέρος υλοποίησης έγινε η ένωση όλων των πιο πάνω κομματιών.

Μετά από την υλοποίηση έγινε η πειραματική αξιολόγηση. Με βάση τα αποτελέσματα της πειραματικής αξιολόγησης έγινε εξαγωγή συμπερασμάτων. Τα πειράματα αυτά έδειξαν αν ο αρχικός στόχος της παρούσας διπλωματικής εργασίας επιτυγχάνεται ή όχι. Επιπρόσθετα, σημαντικό ρόλο για τις μετρήσεις αυτές έχει παίξει και ο χρόνος εκτέλεσης των πειραμάτων. Τα συμπεράσματα τα οποία πάρθηκαν είναι ότι ο

συνολικός αριθμός των ενεργειών που υπάρχουν σε ένα αρχείο ενεργειών μειώνονται με την μέθοδο αυτή.

Η παρούσα διπλωματική εργασία έχει υλοποιηθεί σε γλώσσα προγραμματισμού C. Επίσης, χρησιμοποιήθηκαν τα UNIX του Πανεπιστημίου Κύπρου για την μεταγλώττιση των προγραμμάτων.

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

1.1 Ιστορική αναδρομή

Η τεχνητή νοημοσύνη είναι ο τομέας της επιστήμης των υπολογιστών που ασχολείται με την σχεδίαση και την υλοποίηση προγραμμάτων τα οποία είναι ικανά να μιμηθούν τις ανθρώπινες γνωστικές ικανότητες, εμφανίζοντας έτσι χαρακτηριστικά που αποδίδονται συνήθως σε ανθρώπινη συμπεριφορά. Η ιστορία της τεχνητής νοημοσύνης αρχίζει από το 350 π.Χ. με τους συλλογισμούς του Αριστοτέλη και καταλήγουμε στο σήμερα να έχουμε στην ιστορία της τεχνητής νοημοσύνης αναπτυγμένα και υλοποιημένα αρκετά ευφυή συστήματα. [4]

Ο Προγραμματισμός Δράσης (classical planning) είναι ένας ενεργός τομέας της έρευνας της θεωρητικής επιστήμης των υπολογιστών και της τεχνητής νοημοσύνης. Ο τομέας αυτός της τεχνητής νοημοσύνης έχει αποσπάσει το ενδιαφέρον πολλών ερευνητών και επιστημόνων. Μέσα από αυτό το μεγάλο ενδιαφέρον πολλών ερευνητών έχουν αναπτυχθεί σήμερα πολλές τεχνικές οι οποίες αφορούν το πρόβλημα του προγραμματισμού δράσης.

1.2 Σκοπός της διπλωματικής εργασίας

Στόχος της διπλωματικής εργασίας ήταν η ανάπτυξη μεθόδων για βελτίωση της ποιότητας πλάνων. Δηλαδή, η ανάπτυξη και η υλοποίηση διαφόρων μεθόδων έτσι ώστε να επιτευχθεί η μείωση του συνολικού αριθμού των ενεργειών που υπάρχουν σε ένα αρχείο ενεργειών. Μέσα από τη μελέτη διαφόρων άρθρων και την εξοικείωση με τα εργαλεία τα οποία χρησιμοποιήθηκαν έγινε αντιληπτή η έννοια των προβλημάτων Προγραμματισμού Δράσης.

Στόχος της παρούσας διπλωματικής ήταν η δημιουργία ενός προγράμματος στο οποίο να γίνεται σμίκρυνση του μήκος των ενεργειών στο αρχείο ενεργειών. Για την υλοποίηση του προγράμματος αυτού χρησιμοποιήθηκαν τα εργαλεία LAMA2011 το οποίο είχε σαν αποτέλεσμα ένα αρχείο ενεργειών καθώς επίσης και το εργαλείο SATPLAN2006 με κωδικοποιητή SMP (SAT-MAX-PLAN) το οποίο χρησιμοποιήθηκε για να δίνει ένα αρχείο ενεργειών στην έξοδο μετά από μία επεξεργασία του αρχείου προβλήματος.

1.3 Δομή της διπλωματικής εργασίας

Η παρούσα διπλωματική εργασία μπορεί να θεωρηθεί ότι έχει χωριστεί σε τρεις φάσης μελέτης. Στην πρώτη φάση ήταν η μελέτη διάφορων άρθρων καθώς και βιβλίων για να γίνει η κατανόηση των βασικών εννοιολογικών εννοιών και ορισμών. Η μελέτη ήταν σχετικά με την κατανόηση του προγραμματισμό δράσης. Στη συνέχεια, έγινε εξοικείωση με τα εργαλεία LAMA2011 και SATPLAN2006 με τον κωδικοποιητή SMP (SAT-MAX-PLAN).

Στη δεύτερη φάση έγινε η ανάπτυξη και υλοποίηση των διαφόρων τεχνικών οι οποίες περιγράφονται στο Κεφάλαιο 6 Υλοποίηση. Η φάση αυτή υλοποιήθηκε σε διάφορα μέρη. Στο πρώτο μέρος υλοποιήθηκε το κομμάτι στο οποίο γινόταν η τροποποίηση του αρχείου προβλήματος. Δηλαδή, υπήρχε μια λίστα προσθήκης καταστάσεων και μία λίστα διαγραφής. Η κατάσταση που υπήρχε στη λίστα διαγραφής θα αντικατασταθεί με την ανάλογη κατάσταση που υπάρχει στη λίστα προσθήκης καταστάσεων. Το δεύτερο μέρος, είχε ως σκοπό την συμπίεση του μήκους του αρχείου ενεργειών που είχε σαν αποτέλεσμα το σύστημα LAMA. Μέσα από διάφορες τεχνικές που θα περιγραφούν παρακάτω πέτυχε ο στόχος αυτός. Στο τρίτο μέρος της φάσης αυτής ήταν η ένωση των δύο κομματιών αυτών.

Η τρίτη φάση είχε ως στόχο την πειραματική ανάλυση των τεχνικών που είχαν υλοποιηθεί. Επίσης, στην τεχνική αυτή έγιναν συγκρίσεις μεταξύ των διαφόρων τεχνικών που υλοποιήθηκαν. Μέσα από αυτές τις συγκρίσεις υπήρξε εξαγωγή συμπερασμάτων σχετικά με τις τεχνικές. Στο τέλος, υπάρχουν κάποιες μελλοντικές επεκτάσεις οι οποίες μπορούν να γίνουν στην παρούσα διπλωματική εργασία.

Κεφάλαιο 2

Προγραμματισμός Δράσης (Classical Planning)

2.1 Ορισμός Προγραμματισμού Δράσης

Προγραμματισμός είναι η διαδικασία του να σκέφτεται ο άνθρωπος και να μπορεί να οργανώνει τις δραστηριότητες που απαιτούνται για την επίτευξη του επιθυμητού στόχου.
Ο Προγραμματισμός περιλαμβάνει τη δημιουργία και διατήρηση ενός σχεδίου. Ως εκ τούτου, ο Προγραμματισμός είναι μια θεμελιώδης ιδιότητα του ευφυούς συμπεριφοράς. Αυτή η διαδικασία σκέψης είναι απαραίτητη για τη δημιουργία και τελειοποίηση ενός σχεδίου και για την ενσωμάτωσή του με άλλα σχέδια. Δηλαδή, συνδυάζει την πρόβλεψη των εξελίξεων με την προετοιμασία των σεναρίων για το πώς να αντιδράσουν σε αυτές τις εξελίξεις.

Ο Προγραμματισμός Δράσης (classical planning) είναι ένας ενεργός τομέας της έρευνας της θεωρητικής επιστήμης των υπολογιστών και της τεχνητής νοημοσύνης. Αποτελεί συνεχώς για τους ερευνητές πηγή έμπνευσης και υπάρχει συνεχής και ζωντανή εξέλιξη στην έρευνα του τομέα αυτού. Ο σχεδιασμός ενεργειών είναι πλήρως παρατηρίσιμος, αιτιοκρατικός, πεπερασμένος, ντετερμινιστικός (αποτελέσματα των ενεργειών του πράκτορα, agent), έχει στατικό περιβάλλον (μόνο ο πράκτορας ,agent, μπορεί να αλλάξει το περιβάλλον) και τέλος έχει δυναμικό περιβάλλον (χρόνος, δράσεις, αντικείμενα, αποτελέσματα)[6].

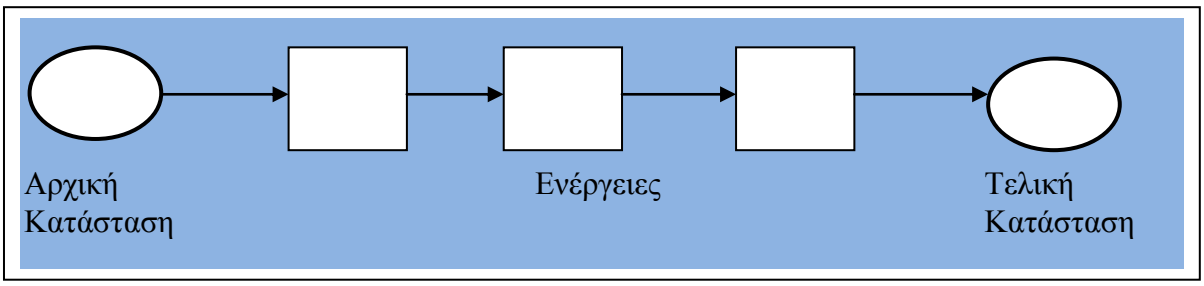
Τα προβλήματα Προγραμματισμού Δράσης (planning problem) είναι τα προβλήματα στα οποία είναι πλήρως γνωστή η τελική κατάσταση και επιθυμείται η εύρεση μιας ακολουθίας από ενέργειες οι οποίες επιτυγχάνουν ένα επιθυμητό στόχο. Η ακολουθία των ενεργειών, οι οποίες είναι η λύση του προβλήματος ονομάζεται πλάνο ενώ το πρόγραμμα που την παράγει ονομάζεται σχεδιαστής.

Στην περίπτωση του Προγραμματισμού Δράσης ισχύει η υπόθεση του κλειστού κόσμου (closed world assumption) δηλαδή, δεν υπάρχει πιθανότητα προσθήκης νέων ή διαγραφής υπαρχόντων αντικειμένων από τον κόσμο του συστήματος.

Παράλληλα, ο Προγραμματισμός Δράσης είναι ο θεμέλιος λίθος για πολλές εφαρμογές. Εφαρμογές όπως τα παιχνίδια, ρομπότ, σε Αυτόνομα Διαστημόπλοια και άλλες τόσες εφαρμογές.

Ένα πρόβλημα Προγραμματισμού Δράσης ορίζεται από τρεις περιγραφές οι οποίες είναι:

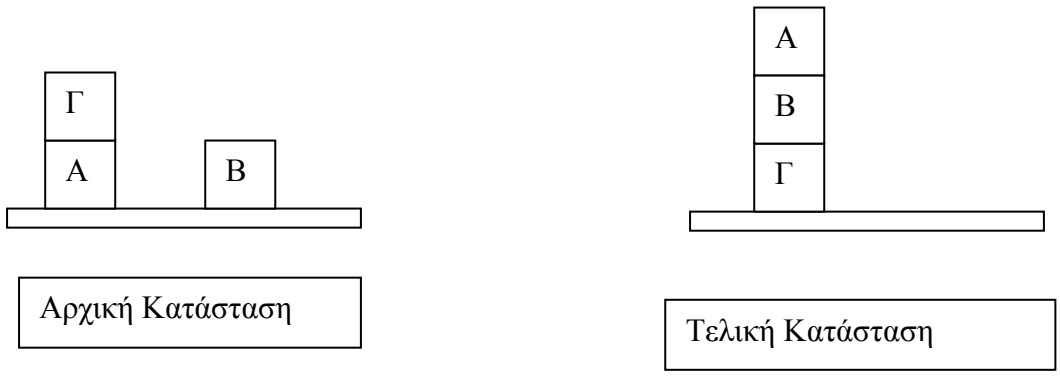
- Αρχική κατάσταση (Initial State)
- Στόχοι (Goals)
- Διαθέσιμες Ενέργειες (Actions)



Σχήμα 2.1 Αναπαράσταση ενός προβλήματος Προγραμματισμού Δράσης

Οι πιο συχνά χρησιμοποιούμενες γλώσσες για την αναπαράσταση προβλημάτων Προγραμματισμού Δράσεων είναι οι γλώσσες STRIPS και PDDL. Το μοντέλο STRIPS(Stanford Research Institute Planning System) έχει προταθεί το 1971 από τους Fikes και Nilsson για ένα μικρό ρομπότ που ονομαζόταν Shakey για τη εκτέλεση απλών ενεργειών. Το μοντέλο είναι ένα απλό μοντέλο που έχει στοιχεία προτασιακής λογικής. Επίσης, το μοντέλο αυτό είναι κατάλληλο για προβλήματα όπου δεν εμφανίζεται αβεβαιότητα. Στο μοντέλο STRIPS οι καταστάσεις ορίζονται σαν σύνολα από συγκεκριμένα γεγονότα ή προτάσεις που αληθεύουν. Πιο κάτω ακολουθεί ένα παράδειγμα 2.1 αναπαράστασης μοντέλου STRIPS που φαίνεται και στο σχήμα 2.2.

Παράδειγμα 2.1:



Σχήμα 2.2 Η αρχική και η τελική κατάσταση για το παράδειγμα «Ο κόσμος των κύβων»

Αρχική κατάσταση:

Block(a) ∧ Block(b) ∧ Block(c) ∧ on(a, table) ∧ on(c, a) ∧ on(b, table) ∧ clear(b) ∧ clear(c)

Τελική κατάσταση:

on(b, c) ∧on(a, b)

Κατηγορήματα:

on(b, c) :ο κύβος b είναι στην κορυφή του κύβου c

block(b) : το b είναι κύβος

clear(b) : έλεγχος αν στην κορυφή του κύβου b δεν υπάρχει άλλος κύβος

table(x) : μεταφορά του κύβου x στο τραπέζι

equal(x, y) : ο κύβος x και ο κύβος y είναι ο ίδιος κύβος

Ενέργειες:

- move (b, c , x): μετακίνησε τον κύβο b από την κορυφή του κύβου x στην κορυφή του κύβου c.

  Προϋποθέσεις (preconditions): on(b, x) ∧ clear(b) ∧ clear(c) ∧ ¬equal(b, x) ∧¬equal(c, b) ∧ ¬equal(c, x) ∧ block(b) ∧ block(c)

Συνέπειες : on(b, c) ∧ clear(x) ∧ ¬on(b, x) ∧ ¬clear(c)

- MoveToTable (b, c): μετακίνησε τον κύβο b στο τραπέζι και μετά από το τραπέζι στην κορυφή του κύβου c
  Προϋποθέσεις : on(b, c) ∧ clear(b) ∧ ¬equal(b, c) ∧block(c) ∧ table(x)
  Συνέπειες : on(b, x) ∧ clear(c) ∧ ¬on(b,c)

Κεφάλαιο 3

Γλώσσες PDDL(Planning Domain Definition Language)

3.1 Ορισμός για τις γλώσσες PDDL

Οι γλώσσες PDDL αποτελούν πρότυπο κωδικοποίησης για τα προβλήματα σχεδιασμού ενεργειών. Προέρχονται από τις αυθεντικές STRIPS γλώσσες σχεδιασμού που αναπτύχθηκαν από τους Fikes και Nilsson το 1971. Αναπτύχθηκαν για πρώτη φορά από τον Drew McDermott και μετέπειτα οι συνάδελφοι του το 1998, επηρεασμένοι από τα STRIPS, συνέχισαν την προσπάθεια του αυτή, κυρίως για να πάρουν μέρος στον διαγωνισμό του International Planning Competition (IPC) το 1998/2000. Στης συνέχεια δημιουργήθηκαν αρκετές εκδόσεις των γλωσσών PDDL.

Με τις γλώσσες PDDL περιγράφονται τέσσερεις τομείς οι οποίοι χρειάζονται για να καθορίσουμε ένα πρόβλημα αναζήτησης. [5]
Χρειάζονται:

- Η αρχική κατάσταση
- Οι καταστάσεις σε κάθε στάδιο
- Το αποτέλεσμα κάθε ενέργειας
- Οι στόχοι της δοκιμής

Ενέργειες στις γλώσσες PDDL:

Κάθε ενέργεια περιγράφεται από ένα σύνολο από ενέργειες οι οποίες διευκρινίζουν τις ενέργειες (Actions(s)) και το αποτέλεσμα (Results(s,a)) που απαιτούνται για να μπορεί να επιλυθεί το πρόβλημα. Επίσης, οι γλώσσες PDDL καθορίζουν το αποτέλεσμα της δράσης σε σχέση με τις αλλαγές που έχουν γίνει. Ο σχεδιασμός ενεργειών εντούτοις, συγκεντρώνει τα προβλήματα που οι περισσότερες ενέργειες του δεν αλλάζουν πολλές καταστάσεις.
Παράδειγμα 3.1:
Action (Fly (p, from, to),
Precond: At (p, from)∧ Plane(p) ∧ Airport(from) ∧Airport(to)
Effect: ¬At (p, from) ∧ At(p, to))

Όπως βλέπουμε στο πιο πάνω παράδειγμα 3.1 οι γλώσσες PDDL αποτελούνται από το όνομα της ενέργειας, μία λίστα από τις μεταβλητές που χρησιμοποιούνται στο σχήμα ενέργειας, τις προϋποθέσεις και τα αποτελέσματα. Μία ενέργεια p μπορεί να εκτελεστεί σε μια κατάσταση s εάν το s προϋποθέτει την προϋπόθεση a. Επιπρόσθετα, οι προϋποθέσεις προσδιορίζουν τις καταστάσεις στις οποίες η δράση μπορεί να εκτελεστεί. Το αποτέλεσμα προσδιορίζει το αποτέλεσμα της ενέργειας που έχει εκτελεστεί. Οι προϋποθέσεις και τα αποτελέσματα μπορεί να αποτελούνται από θετικούς και αρνητικούς λεκτικούς όρους. Επίσης, στις γλώσσες PDDL μπορεί να υπάρχουν η λίστα προσθήκης στην οποία προστίθενται ενέργειες και η λίστα διαγραφής στην οποία διαγράφονται ενέργειες.

Για τις γλώσσες PDDL υπάρχει ένα αρχείο που ονομάζεται planning domain στο οποίο είναι αποθηκευμένες όλες οι ενέργειες του προβλήματος. Επίσης, υπάρχει ένα αρχείο problem στο οποίο ένα συγκεκριμένο πρόβλημα ορίζεται από μία αρχική κατάσταση και ένα στόχο. Ο στόχος είναι ένας συνδυασμός από λεκτικούς όρους που μπορεί να είναι είτε αρνητικοί είτε θετικοί και μπορεί να περιέχει μεταβλητές, όπου αυτές οι μεταβλητές αντιμετωπίζονται ως υπαρξιακοί ποσοδείκτες. Για παράδειγμα, για τον στόχο At (p, LAR)∧ Plane(p) τότε πρέπει να υπάρχει ένα οποιοδήποτε αεροπλάνο για το αεροδρόμιο LAR.

3.2 Παραδείγματα

Παράδειγμα 3.2 **Αερομεταφοράς Φορτίων:**

Αρχική Κατάσταση (Στο(C1, SFO) ^ Στο(C2, JFK) ^ Στο(P1, SFO) ^ Στο(P2, JFK)^ Φορτίο(C1) ^ Φορτίο(C2) ^ Αεροπλάνο(P1) ^ Αεροπλάνο(P2)^ Αεροδρόμιο(JFK) ^ Αεροδρόμιο(SFO))
Goal (Στο (C1, JFK) ^ Στο(C2, SFO))

Ενέργεια (Φόρτωμα (c, p, a),
PRECOND: Στο(c, a) ^ Στο(p, a) ^ Φορτίο(c) ^ Αεροπλάνο(p) ^ Αεροδρόμιο(a)
EFFECT: Στο(c, a) ^ Μέσα(c, p))

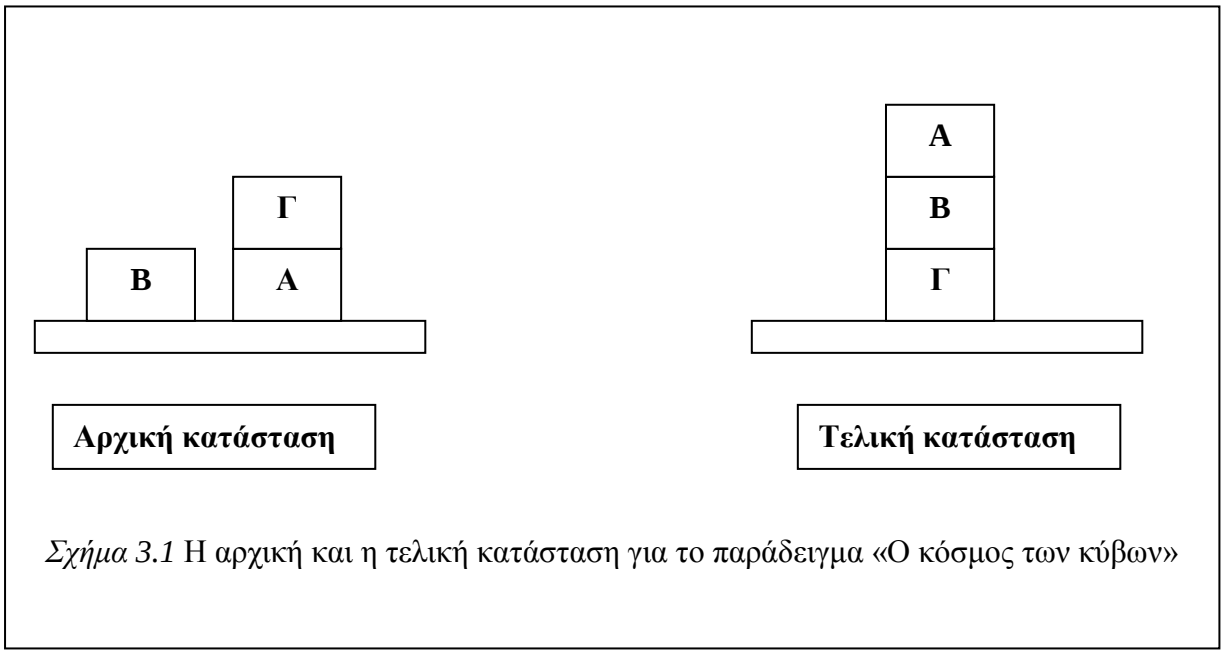
Ενέργεια (Ξεφόρτωμα (c, p, a),
PRECOND: Μέσα(c, p) ^ Στο (p, a) ^ Φορτίο(c) ^ Αεροπλάνο(p) ^ Αεροδρόμιο(a)
EFFECT: Στο(c, a) ^ Μέσα(c, p))

Ενέργεια (Πτήση(p, from, to),
PRECOND: Στο(p, from) ^ Αεροπλάνο(p) ^ Αεροδρόμιο(from) ^ Αεροδρόμιο(to)
EFFECT: Στο(p, from) ^ Στο(p, to))

Το πρόβλημα αυτό είναι ένα πρόβλημα αερομεταφοράς φορτίου στο οποίο επιθυμείται η μεταφορά των φορτίων από το ένα μέρος στο άλλος. Οι ενέργειες που γίνονται σε αυτό το πρόβλημα είναι το φόρτωμα ενός φορτίου, το ξεφόρτωμα ενός φορτίου και οι πτήσεις. Επίσης, υπάρχει το κατηγόρημα *Μέσα (c,p)* που σημαίνει ότι το φορτίο c είναι μέσα στο αεροπλάνο p και το κατηγόρημα *Στο(x,a)* που σημαίνει ότι το αντικείμενο x (που είναι το αεροπλάνο p μέσα στο οποίο βρίσκεται το φορτίο c) είναι μέσα στο αεροδρόμιο a. Όταν ένα αεροπλάνο ταξιδεύει από ένα αεροδρόμιο σε άλλο τότε συνεπάγεται ότι και το φορτίο μεταφέρεται μαζί του. Επίσης, όταν το αεροπλάνο φθάσει στο αεροδρόμιο τότε το φορτίο παύει να είναι *Στο αεροπλάνο* και γίνεται το φορτίο *Στο νέο αεροδρόμιο* όπου εκεί ξεφορτώνεται. Άρα, ουσιαστικά ο όρος *Στο(x,a)* αλλάζει ανάλογα με την τοποθεσία που βρίσκεται το φορτίο.

Η αρχική κατάσταση, δηλαδή η κατάσταση που είναι αρχικά τα φορτία είναι: (Στο(C1, SFO) ^ Στο(C2, JFK) ^ Στο(P1, SFO) ^ Στο(P2, JFK)^ Φορτίο(C1) ^ Φορτίο(C2) ^ Αεροπλάνο(P1) ^ Αεροπλάνο(P2)^ Αεροδρόμιο(JFK) ^ Αεροδρόμιο(SFO)) και η τελική κατάσταση στην οποία τα φορτία επιθυμείται να μεταφερθούν είναι: Goal (Στο (C1, JFK) ^ Στο(C2, SFO))
Δηλαδή, επιθυμείται τα φορτία C1 και C2 να είναι στις τοποθεσίες JFK και SFO αντίστοιχα. Το φορτίο C1 αρχικά ήταν στην τοποθεσία SFO και το φορτίο C2 ήταν στην τοποθεσία JFK. Η λύση για το πρόβλημα αυτό είναι:
[Φόρτωμα (C1, P1, SFO), Πτήση (P1, SFO, JFK), Ξεφόρτωμα (C1, P1, JFK),
Φόρτωμα (C2, P2, JFK), Πτήση(P2, JFK, SFO), Ξεφόρτωμα (C2, P2, SFO)]

***Παράδειγμα 3.3*: Ο κόσμος των κύβων**



Σχήμα 3.1 Η αρχική και η τελική κατάσταση για το παράδειγμα «Ο κόσμος των κύβων»

Στο πρόβλημα αυτό οι κύβοι μπορούν να τοποθετούνται πάνω σε ένα κύβο ή περισσότερους και να γίνονται μια στοίβα αλλά μόνο ένας κύβος μπορεί να είναι στην κορυφή ενός άλλου κύβου ή να είναι τοποθετημένοι πάνω στο τραπέζι. Στόχος για το πρόβλημα αυτό είναι η μετακίνηση των διαφόρων κύβων και η κατάληξη τους από την αρχική κατάσταση στην τελική κατάσταση. Όπως παρατηρείται στο πιο πάνω σχήμα 3.1 υπάρχει η αρχική κατάσταση στην οποία ο κύβος Β και ο κύβος Α είναι πάνω στο τραπέζι και ο κύβος Γ είναι πάνω στον κύβο Α. Η τελική κατάσταση η οποία επιθυμείται είναι ο κύβος Α να είναι πάνω στον κύβο Β, ο κύβος Β να είναι πάνω στον κύβο Γ και ο κύβος Γ είναι πάνω στο τραπέζι.

Init (On (A, Table) ^ On(B, Table) ^ On(C, A)^ Block(A) ^ Block(B) ^ Block (C) ^ Clear(B) ^ Clear(C))
Goal (On (A, B) ^ On(B, C))

Action (Move (b, x, y),
PRECOND: On (b, x) ^ Clear (b) ^ Clear(y) ^ Block(b) ^ Block(y) ^ (b ≠ x) ^ (b ≠ y) ^ (x ≠ y)
EFFECT: On (b, y) ^ Clear(x) ^ ¬ On(b, x) ^ ¬ Clear(y))

Action (MoveToTable(b, x);
PRECOND: On(b, x) ^ Clear(b) ^ Block(b) ^ (b≠x),
EFFECT: On(b, Table) ^¬ Clear(x) ^ ¬On(b, x))

Στο πρόβλημα αυτό υπάρχει η ενέργεια *On(x, y)* για την οποία δηλώνεται ότι ο κύβος x βρίσκεται πάνω στο y και ότι το y είναι ένας άλλος κύβος ή το τραπέζι. Επίσης, υπάρχει η ενέργεια *Move(b,x,y)* στην οποία ο κύβος b τοποθετείται στην κορυφή του αντικειμένου y από την κορυφή του αντικειμένου x. Η ενέργεια αυτή γίνεται εάν στην κορυφή του b κύβου δεν υπάρχει άλλος κύβος και στην κορυφή του κύβου y δεν υπάρχει οποιοσδήποτε άλλος κύβος. Υπάρχει όμως μια προϋπόθεση, στην οποία για την μετακίνηση ενός κύβου b στην κορυφή του κύβου x δεν υπάρχει άλλος κύβος στην κορυφή του x. Επειδή οι γλώσσες PDDL δεν επιτρέπουν ποσοδείκτες έτσι υπάρχει ένα κατηγόρημα *Clear(x)* που είναι αληθείς αν δεν υπάρχει οποιοσδήποτε κύβος στην κορυφή του κύβου x. Βάζοντας αυτό το κατηγόρημα επιτρέπεται με αυτό τον τρόπο ο έλεγχος αν μπορεί να μετακινηθεί ο οποιοσδήποτε κύβος στην κορυφή του κύβου x. Υπάρχει ακόμη μία ενέργεια *MoveToTable(b, x)* η οποία ενεργεί με τον ίδιο τρόπο με την *Move*. Αλλά, δεν χρειάζεται να υπάρχει η προϋπόθεση *Clear(Table)* γιατί είναι πάντα αληθής η συνθήκη αυτή και δεν χρειάζεται ο έλεγχος αν δεν υπάρχει άλλος κύβος στο τραπέζι γιατί θεωρητικά το τραπέζι είναι πολύ μεγάλο και μπορούν να τοποθετηθούν πάνω σε αυτό πάρα πολλοί κύβοι. Αυτό που γίνετε όταν έχουμε αυτή την ενέργεια είναι ότι μετακινείτε ο κύβος b στο τραπέζι και στη συνέχεια ελέγχεται ο αν είναι αληθής η συνθήκη *Clear(x)* και αν είναι αληθής τότε μετακινείται στο κύβο x.

Κεφάλαιο 4

Το Σύστημα LAMA και το σύστημα SMP

Τα προβλήματα Προγραμματισμού Δράσης (Classical Planning) έχουν συνήθως πολυπλοκότητα PSPACE καθώς επίσης υποφέρουν από ανακρίβειες . Όμως τα προβλήματα του πραγματικού κόσμου δεν χρειάζονται τόση μεγάλη πολυπλοκότητα για να μπορούν να επιλυθούν αφού μπορούν να επιλυθούν σε μικρότερο χρονικό διάστημα και με μικρότερη πολυπλοκότητα.

Μεγάλος αριθμός επιστημόνων και ερευνητών έδειξαν το ενδιαφέρον τους για να μελετήσουν και να αναπτύξουν διάφορες μεθόδους για την επίλυση προβλημάτων Προγραμματισμού Δράσης. Σκοπός των επιστημόνων ήταν η δημιουργία εργαλείων τα οποία θα έλυναν τα προβλήματα αυτά με μικρότερη πολυπλοκότητα καθώς επίσης να είναι αποδοτικά και αξιόπιστα ως προς την λύση τους.

4.1 Το σύστημα LAMA

Η έκδοση του συστήματος LAMA που χρησιμοποιήθηκε σε αυτή την διπλωματική είναι μία έκδοση η οποία έχει λάβει μέρος στο διαγωνισμό International Planning Competition(IPC) το 2011. Για το πρόγραμμα αυτό έχει χρησιμοποιηθεί κώδικας από τον κώδικα των ερευνητών Silvia Richter και Matthias Westphal του 2008 καθώς επίσης και κώδικας από το Fast Downward Planner των Silvia Richter και Malte Helmert.

Ο αυτοματοποιημένος σχεδιασμός και προγραμματισμός είναι ένας κλάδος της τεχνητής νοημοσύνης που αφορά την υλοποίηση των στρατηγικών ή των ενεργειών. Χρησιμοποιείται

για την υλοποίηση ευφυών πρακτόρων και αυτόνομων ρομπότ. Στα γνωστά περιβάλλοντα με διαθέσιμα μοντέλα, ο σχεδιασμός μπορεί να γίνει χωρίς σύνδεση. Επίσης, λύσεις μπορούν να βρεθούν και να αξιολογηθούν πριν από την εκτέλεση. Αντίθετα, σε δυναμικά άγνωστα περιβάλλοντα, η στρατηγική θα πρέπει συχνά να αναθεωρηθεί σε απευθείας σύνδεση. Τα μοντέλα και οι πολιτικές πρέπει να προσαρμόζονται. Οι λύσεις συνήθως καταφεύγουν σε επαναληπτική δοκιμή.

Το σύστημα LAMA είναι μία προσέγγιση που χρησιμοποιείται για την επίλυση προβλημάτων προγραμματισμού δράσης. Η προσέγγιση που χρησιμοποιεί το πρόγραμμα αυτό είναι η ευρετική αναζήτηση. Ο LAMA υπολογίζει στην αρχή μια γρήγορη λύση με τη χρήση ενός αποδοτικού αλγορίθμου και μετά γίνεται αναζήτηση για μικρότερου μήκους πλάνα καλώντας διαδοχικές εκτελέσεις του A* αλγορίθμου με μειωμένα τα βάρη.

4.2 Το σύστημα SATPLAN και ο κωδικοποιητής SMP

Οι επιστήμονες Henry Kautz και Bart Selman το 1992 ανέπτυξαν μια ιδέα για την επίλυση προβλημάτων προγραμματισμού δράσης, οι οποίοι είχαν προτείνει τον μετασχηματισμό του προβλήματος Προγραμματισμού Δράσης σε προβλήματα Προτασιακής Λογικής[1]. Το 1996, σε μία ανανεωμένη έκδοση της έρευνας των επιστημόνων αυτών δημιούργησαν το σύστημα SATPLAN το οποίο στηρίζεται στην προτασιακή λογική. Το σύστημα αυτό το 2004 έχει πάρει το πρώτο βραβείο στο συνέδριο ICAPS. Επίσης, το 2006 δημιουργήθηκε μια ανανεωμένη έκδοση του συστήματος SATPLAN2004 στην οποία έγιναν βελτιστοποιήσεις για την πιο αποδοτική λειτουργία της έκδοσης του 2006.

4.2.1 Τρόπος λειτουργίας του συστήματος SATPLAN2006:

Το σύστημα SATPLAN2006 δέχεται στην είσοδο προβλήματα που ανήκουν στο υποσύνολο STRIPS της γλώσσας PDDL και βρίσκει μία λύση η οποία θα έχει το μικρότερο μήκος. Ο συνολικός αριθμός των βημάτων, που είναι η λύση του προβλήματος, είναι εγγυημένο ότι θα είναι ο μικρότερος αριθμός χρονικών βημάτων που χρειάζεται ένα πρόβλημα για να επιλυθεί[1]. Για την επίλυση των προβλημάτων αυτών το σύστημα SATPLAN2006 χρησιμοποιεί προτασιακούς επιλυτές. Επίσης, όσο πιο αποδοτικός είναι ο προτασιακός

επιλυτής τόσο πιο αποδοτική θα είναι και η επίλυση του προβλήματος του προγραμματισμού δράσης.

Το σύστημα SATPLAN2006 αρχικά κατασκευάζει ένα γράφο GraphPlan, ο οποίος είναι ένας γράφος ενεργειών που φτάνει μέχρι κάποιο χρονικό επίπεδο k. Στη συνέχεια, μεταφράζει τους περιορισμούς οι οποίοι προκύπτουν σε Συζευκτική Κανονική Μορφή(CNF). Η Συζευκτική Κανονική Μορφή είναι η σύζευξη λεκτικών (clauses) και μπορούν να περιέχουν μεταβλητές καθολικά ποσοτικοποιημένες. Ένα παράδειγμα από μια πρόταση «όλοι όσοι αγαπούν όλα τα ζώα αγαπώνται από κάποιον» η μετατροπή σε CNF μορφή είναι $\forall x [\forall y \text{ Ζώο}(y) \Rightarrow \text{Αγαπά}(x, y)] \Rightarrow [\exists y \text{ Αγαπά}(y, x)]$.[7] Στη συνέχεια, η CNF μορφή δίνεται ως είσοδος σε ένα επιλυτή που προσπαθεί να βρει μια ικανοποιητική ανάθεση στην οποία όλες οι προτάσεις να γίνονται αληθής. Εάν υπάρχει αυτή η ανάθεση τότε μας επιστρέφει την λύση του προβλήματος αλλιώς υπάρχει ενημέρωση ότι δεν υπάρχει λύση.

Ακολουθεί ο αλγόριθμος ο οποίος χρησιμοποιεί το σύστημα SATPLAN2006:

```
function SATPlan (init, actions, goal, Tmax) returns solution or failure

inputs: init, actions, goal

Tmax, an upper limit for plan length

        for t = 0 to Tmax do
cnf  ←  Translate-To-Sat(init, actions, goal, t)
model ← SAT-Solver(cnf)

if model is not null then
        return Extract-Solution(model)

return failure
```

Σε αυτή την Διπλωματική Εργασία, χρησιμοποιείται ο κωδικοποιητής SAT-MAX-PLAN(SMP) σε Συζευκτική Κανονική Μορφή (CNF) ο οποίος είναι ένας κωδικοποιητής των περιορισμών των Προβλημάτων Δράσεων που θεωρείται ότι είναι καλύτερος από άλλους κωδικοποιητές όπως είναι ο κωδικοποιητής BLACKBOX και SATPLAN2006. Επίσης, ο SMP θεωρείται ότι επιτυγχάνει μεγαλύτερη διάδοση περιορισμών από ότι τα άλλα

μοντέλα[2]. Στόχος του SMP είναι η μετάφραση των περιορισμών σε Συζευκτική Κανονική Μορφή (CNF) είναι το πρόβλημα να γίνει πρόβλημα Ικανοποιησιμότητας για να μπορεί να επιλυθεί με ένα επιλυτή.

Ένας προτασιακός επιλυτής έχει ως στόχο την επίλυση των Προβλημάτων Ικανοποιησιμότητας. Το πρόβλημα της Προτασιακής Ικανοποιησιμότητας είναι να μπορεί να εξευρεθεί ανάθεση τιμών στις μεταβλητές μια πρότασης ώστε αυτή η πρόταση να είναι αληθής. Αν για αυτή την πρόταση δεν μπορεί να εξευρεθεί κάποια ανάθεση τότε η πρόταση αυτή είναι ψευδής. Το πρόβλημα ελέγχου της Ικανοποιησιμότητας λογικών εκφράσεων (γνωστό ως SAT) είναι ένα από τα σημαντικότερα θέματα που απασχολούν τους ερευνητές που εστιάζονται στο χώρο της Τεχνητής Νοημοσύνης. Παράλληλα, αποτελεί ένα από τα πλέον δημοφιλή προβλήματα της τάξης NP-Πλήρης. Η επίλυσή του έχει μάλιστα ποικίλες εφαρμογές σε διάφορα πρακτικά προβλήματα, όπως για παράδειγμα στο σχεδιασμό σε προβλήματα Τεχνητής Νοημοσύνης (AI Planning).

4.2.2 Προτασιακή Λογική

Η προτασιακή Λογική ορίζεται από τους επιστήμονες ως ένας φορμαλισμός κάποιων μορφών συλλογιστικής. Κάθε γεγονός του πραγματικού κόσμου αναπαριστάται με μία λογική πρόταση. Αυτή η λογική πρόταση μπορεί να χαρακτηριστεί ως ψευδής ή ως αληθής. Στο σύστημα SATPLAN οι επιλυτές δέχονται ως είσοδο τους περιορισμούς του προβλήματος σε Συζευκτική Κανονική Μορφή (CNF). Στην προτασιακή λογική μια πρόταση είναι σε Συζευκτική Κανονική Μορφή (CNF) όταν οι προτάσεις αποτελούνται από συζεύξεις διαζεύξεων.

Κεφάλαιο 5

Αλγόριθμοι για το σχεδιασμό με Αναζήτηση στο Χώρο των Καταστάσεων και Σχεδιασμός Βασισμένος στο Γράφο

Στην τεχνητή νοημοσύνη ο σχεδιασμός του χώρου καταστάσεων είναι μια διαδικασία που χρησιμοποιείται στο σχεδιασμό προγραμμάτων για την αναζήτηση δεδομένων ή για την εξεύρεση λύσεων σε προβλήματα. Σε έναν αλγόριθμο ο χώρος καταστάσεων είναι ένας συλλογικός όρος για όλα τα δεδομένα που πρέπει να αναζητηθούν. Ομοίως, προγράμματα τεχνητής νοημοσύνης χρησιμοποιούν συχνά μια διαδικασία αναζήτησης μέσα από ένα πεπερασμένο σύνολο πιθανών διαδικασιών για την επίτευξη ενός στόχου, ούτως ώστε να βρεθεί μια διαδικασία ή η καλύτερη διαδικασία για να επιτευχθεί ο στόχος. Το σύνολο των πιθανών λύσεων που θα αναζητηθούν ονομάζεται χώρος καταστάσεων. Χώρος καταστάσεων είναι η διαδικασία να αποφασιστεί από το πρόγραμμα, ποια μέρη του χώρου καταστάσεων θα ψάξει και σε ποια σειρά.

5.1 Προς τα εμπρός προέλαση στον χώρο των καταστάσεων (Forward state-space search)

Προς τα εμπρός προέλαση είναι ένας αλγόριθμος που ψάχνει προς τα εμπρός, έτσι όπως φαίνεται στο σχήμα 5.1, από την αρχική κατάσταση του κόσμου για να προσπαθήσουμε να βρούμε μια κατάσταση που να ικανοποιεί τον στόχο μας. Ξεκινώντας από την αρχική κατάσταση και χρησιμοποιώντας τις ενέργειες του προβλήματος για την αναζήτηση προς τα εμπρός για ένα μέλος του συνόλου των στόχων.

Ακολουθεί ο αλγόριθμος που χρησιμοποιείται για τον αλγόριθμο Προς τα εμπρός:

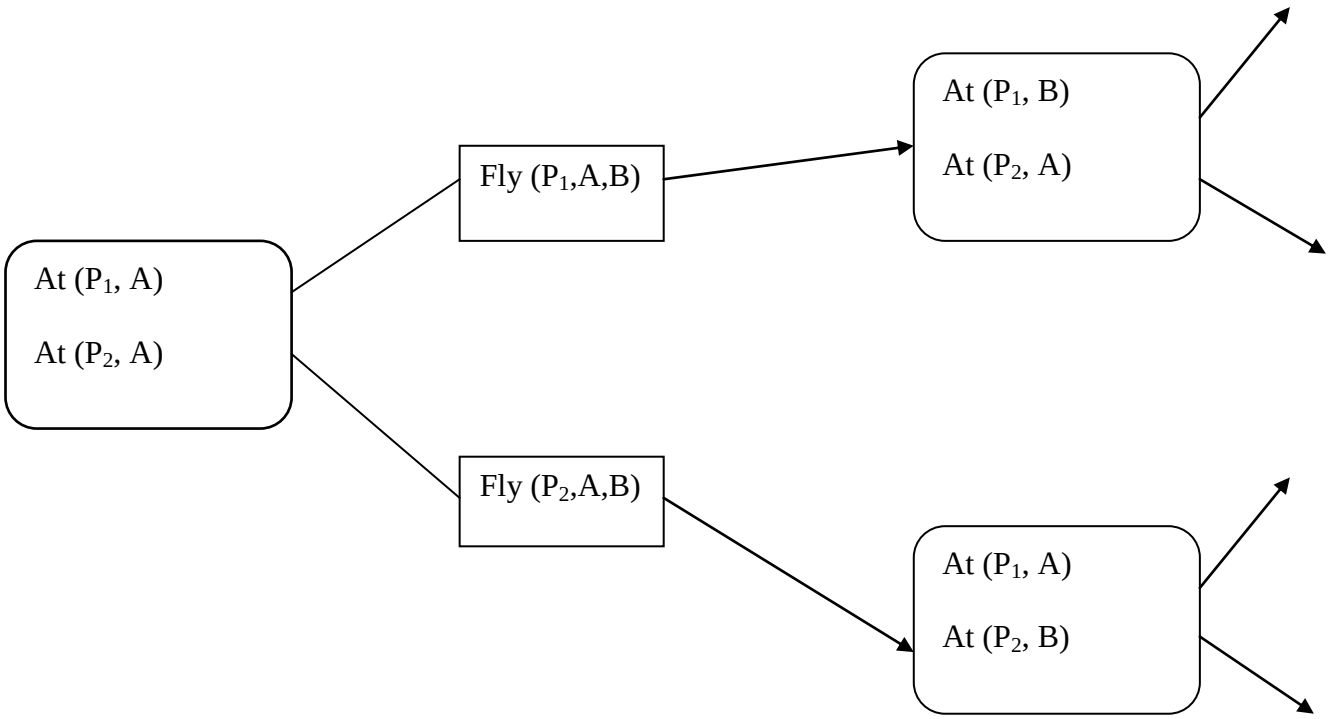
```
Forward-search(O, s0, g)

s = S0
 P = the empty plan

 loop
    if s satisfies g then return P
    applicable = {a | a is a ground instance of an operator in O,and precond(a) is true
in s}
    if applicable = ∅ then return failure
    nondeterministically choose an action a from applicable
    s = γ(s,a)
    P = P.a
```

Αρχίζοντας από την αρχική κατάσταση εφαρμόζονται οι ενέργειες που θα δημιουργήσουν νέες καταστάσεις. Έστω μια αρχική κατάσταση s, ένα σύνολο στόχων g και ένα σύνολο ενεργειών O. Επιλέγεται μια ενέργεια a της οποίας οι προϋποθέσεις της περιέχονται στη αρχική κατάσταση s. Μετά από την εγγραφή της ενέργειας προκύπτει μια νέα κατάσταση S όπου S=(I-Del(a)) U Add(a). Η διαδικασία εφαρμόζεται επαναληπτικά μέχρι να βρεθεί μια κατάσταση που να είναι υποσύνολο των στόχων.

Ιδιότητες για Προς τα εμπρός αναζήτηση:

- Προς τα εμπρός αναζήτηση είναι υγιείς όταν για κάθε μονοπάτι που επιστρέφεται από οποιαδήποτε μη ντετερμινιστικά ίχνη είναι εγγυημένο ότι είναι μια λύση το μονοπάτι αυτό.

- Προς τα εμπρός αναζήτηση είναι πλήρης, αν υπάρχει λύση για την αναζήτηση αυτή τότε τουλάχιστον ένα από τα μη ντετερμινιστικά ίχνη θα επιστραφούν σαν λύση για το πρόβλημα.



Σχήμα 5.1 Προς τα εμπρός αναζήτηση

Για το παράδειγμα αερομεταφοράς φορτίων γίνεται η υπόθεση ότι υπάρχουν 10 αεροδρόμια, όπου στο κάθε αεροδρόμιο υπάρχουν 5 αεροπλάνα και 20 φορτία τα οποία χρειάζονται μεταφορά. Ο στόχος είναι να μεταφερθεί το φορτίο από το αεροδρόμιο Α στο αεροδρόμιο Β. Υπάρχει ένας απλός τρόπος για να μπορεί να λυθεί το πρόβλημα ο οποίος είναι να φορτωθεί το φορτίο σε ένα αεροπλάνο Α από το αεροδρόμιο Α και να γίνει πτήση στο αεροδρόμιο Β και να ξεφορτωθεί το φορτίο. Όμως, για να βρεθεί η λύση ίσως είναι δύσκολο γιατί η διακλάδωση που θα έχουν οι καταστάσεις είναι πολύ μεγάλη. Το κάθε ένα από το 50 (5 αεροπλάνα το κάθε αεροδρομιο*10 αεροδρόμια) αεροπλάνα μπορούν να κάνουν πτήση στα 9 αεροδρόμια (δεν μπορεί να πετάξει στο δικό του αεροδρόμιο) και 200 πακέτα πρέπει να φορτωθούν ή να ξεφορτωθούν. Σε κάθε κατάσταση επομένως, υπάρχουν τουλάχιστον 450 ενέργειες και 10450 μέγιστος αριθμός ενεργειών.

5.2 Προς τα πίσω εξέλιξη στον χώρο των καταστάσεων (Backward state-space search)

Προς τα πίσω εξέλιξη στον χώρο των καταστάσεων πρόκειται για μια αναζήτηση προς την αντίθετη κατεύθυνση, έτσι όπως φαίνεται στο σχήμα 5.2, στην οποία η αναζήτηση ξεκινά με την κατάσταση στόχου και εφαρμόζονται οι ενέργειες μέχρι να βρεθεί μια ακολουθία από βήματα στην οποία φτάνει στην αρχική κατάσταση. Ονομάζονται σχετικές –καταστάσεις γιατί εξετάζονται μόνο οι καταστάσεις οι οποίες έχουν σχέση με τον στόχο.

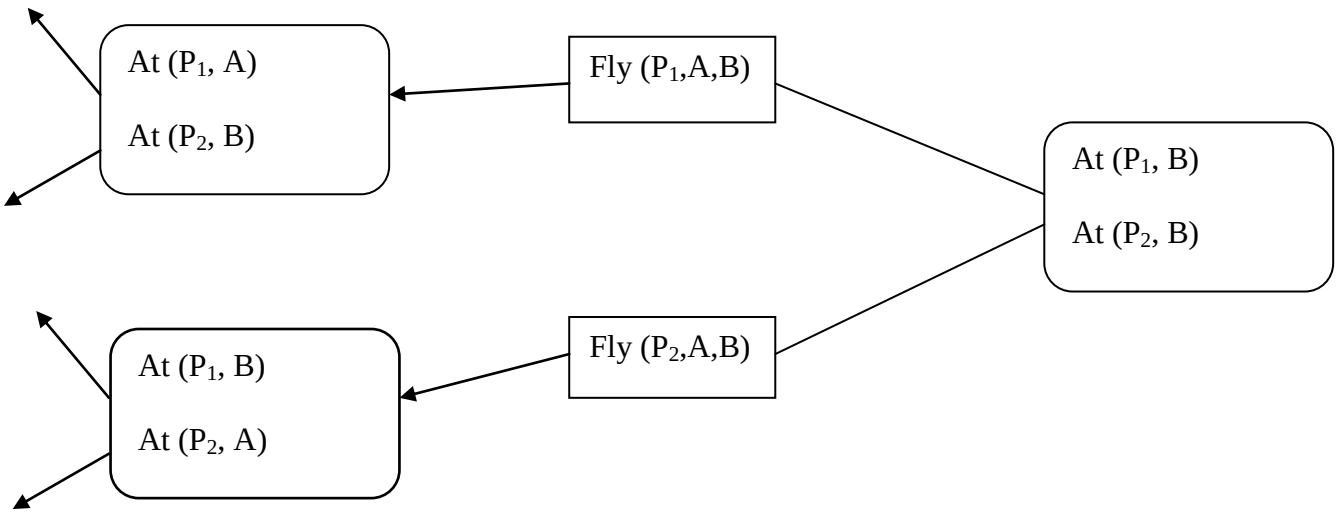
Όπως έχει προαναφερθεί, η αναζήτηση προς τα πίσω γίνεται μόνο όταν είναι γνωστό πως θα γίνει η διαδρομή από την κατάσταση περιγραφής στην προκάτοχο της κατάσταση περιγραφής. Είναι δύσκολο σε μερικά παραδείγματα να γίνει αναζήτηση προς τα πίσω. Όμως οι PDDL γλώσσες έχουν σχεδιαστεί για να κάνουν πιο εύκολη την διαδικασία αυτή. Σε αυτές τις γλώσσες, δίνεται μία περιγραφή του στόχου g και μια ενέργεια a. Η οπισθοδρόμηση από το g στο g' δίνεται με τον όρο: g' = (g− ADD (a))∪ PRECOND (a).

Ακολουθεί ο αλγόριθμος Προς τα πίσω:

```
Backward-search(O, s0, g)

P = the empty plan

 Loop

    if s0 satisfies g then return P
    relevant = {a |a is a ground instance of an operator in O that is relevant for g}

    if relevant = ∅ then return failure
    nondeterministically choose an action a from relevant
    P = a.P
    s = γ^-1(s,a)
```

Στη περίπτωση αυτή η αναζήτηση ξεκινά από τους στόχους προς την αρχική κατάσταση. Έστω μια αρχική κατάσταση I, ένα σύνολο στόχων g και ένα σύνολο ενεργειών. Επιλέγεται μια ενέργεια a τέτοια ώστε κανένα από τα γεγονότα που αυτή διαγράφει να μην εμφανίζεται

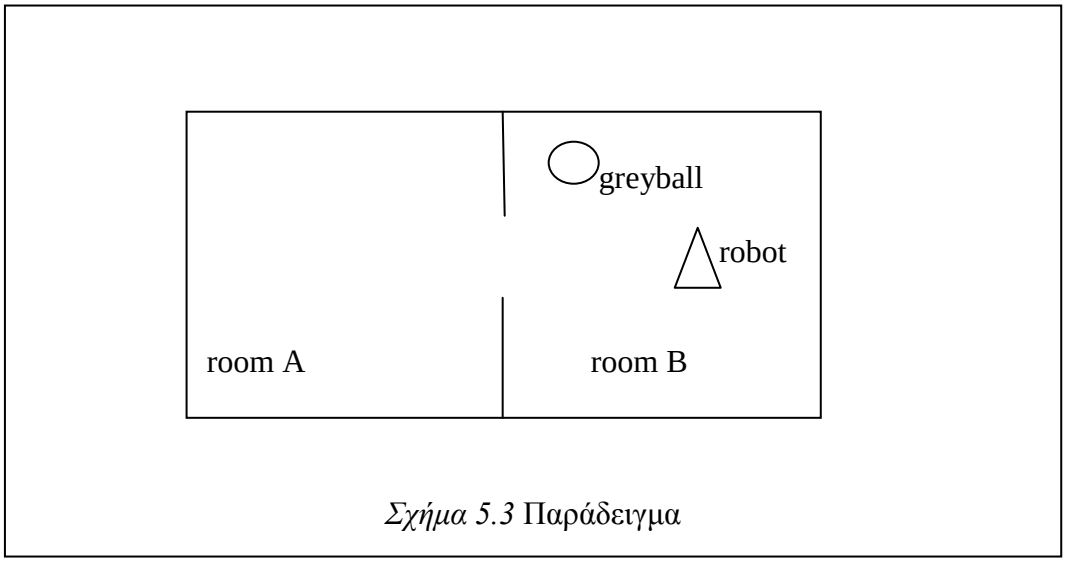
στην τελική κατάσταση, ενώ πρέπει να εμφανίζεται τουλάχιστον ένα από τα γεγονότα που αυτή προσθέτει, $Del(a) \cap G = \emptyset$ και $Add(a) \cap G \neq \emptyset$. Το σύνολο των στόχων αναθεωρείται και γίνεται $g' = (g - ADD(a)) \cup PRECOND(a)$. Η διαδικασία εφαρμόζεται επαναληπτικά στο νέο σύνολο στόχων μέχρι να βρεθεί ένα σύνολο γεγονότων που να είναι υποσύνολο της αρχικής κατάστασης.



Σχήμα 5.2 Προς τα πίσω αναζήτηση

Μέσω των συνόλων των σχετικών καταστάσεων, η αναζήτηση ξεκινά από το σύνολο των καταστάσεων που αντιπροσωπεύουν το στόχο και χρησιμοποιώντας το αντίστροφο των δράσεων για να γίνει αναζήτηση των δράσεων προς τα πίσω για την αρχική ενέργεια.

Παράδειγμα 5.1:



Σχήμα 5.3 Παράδειγμα

Αρχική κατάσταση:

START={robot(robby) ∧ room(roomA) ∧ room(roomB) ∧ ball(greyball) ∧ at(robby,roomB) ∧at(greyball,roomB) ∧free(robby)}

Τελική κατάσταση:

END={at(greyball, roomA)}

Ενέργειες:

- move(R, X, Y)
- pick_ball (R,B,X)
- drop_ball (R,B,X)

Επίλυση με Προς τα Εμπρός Αναζήτηση:

1. Αν precond(A_1) είναι υποσύνολο του START και precond(A_2) είναι υποσύνολο του START. Μπορούν να εφαρμοστούν οι δύο ενέργειες A_1=move(robby, roomB, roomA) και A_2=pick_ball(robby, greyball, roomB).

Αν ο αλγόριθμος αναζήτησης επιλέξει την ενέργεια A_1 τότε Sa=START-del(A_1)U Add($A_{1)=}${ robot(robby)∧room(roomA) ∧room(roomB) ∧ball(greyball) ∧at(robby,roomA) ∧at(greyball,roomB) ∧free(robby)}

2. Στην κατάσταση Sa μπορεί να εφαρμοστεί μόνο η ενέργεια A_3=move(robby,roomA,roomB) από την οποία προκύπτει η κατάσταση Sb={ robot(robby)∧room(roomA) ∧room(roomB) ∧ball(greyball) ∧at(robby,roomB) ∧at(greyball,roomB) ∧free(robby)} }
3. Ο αλγόριθμος επιλέγει την ενέργεια A_2 και προκύπτει η S_c= START-del(A_2)U Add($A_{2)=}${robot(robby)∧room(roomA)∧room(roomB)∧ball(greyball)∧ at(robby,roomA) ∧at(grayball,roomA) ∧has(robby,greyball)}
   Επαναλαμβάνεται η διαδικασία αυτή μέχρι να βρεθεί κατάσταση S_f για την οποία ισχύει END υποσύνολο S_f.

Επίλυση με Προς τα Πίσω Αναζήτηση:

1. Η ενέργεια που προσθέτει το στόχο at(greyball, roomA) είναι η A_1=drop_ball(robby,greyball,roomA). Επομένως, προκύπτει το νέο σύνολο στόχων Ga= precond(A_1)UEND-Add(A_1) ={at(robby,roomA), has(robby, greyball)}
2. Στο σύνολο Ga μπορεί να εφαρμοστούν αντίστροφα οι ενέργειες: A_2=move(robby,roomB,roomA),A_3=pick_ball(robby,greyball,roomB),A_4=pick_ball(robby,greyball,roomA). Έστω, ότι επιλέγεται η ενέργεια A_3 οπότε G_b={ at(robby,roomA) , at(robby,roomB), at(greyball,roomB), free(robby)}. Προκύπτει όμως ότι το G_b δεν είναι έγκυρο καθώς τα γεγονότα: at(robby,roomA) και at(robby,roomB) είναι ασύμβατα μεταξύ τους και ο αλγόριθμος οπισθοδρομεί στο προηγούμενο βήμα.
3. Έστω, ότι επιλέγεται η ενέργεια A_2 οπότε G_c={ at(robby,roomB) ,has(robby,greyball)}. Επαναλαμβάνεται η ίδια διαδικασία μέχρι να βρεθεί το σύνολο στόχων το οποίο να περιέχει όλα τα δυναμικά γεγονότα της αρχικής κατάστασης.

5.3 Σύγκριση των δύο αλγορίθμων

Οι δύο αλγόριθμοι έχουν την ίδια πολυπλοκότητα και θα εκτελούν τον ίδιο αριθμό επαναλήψεων. Όμως, σε μια πραγματική υλοποίηση δεν υπάρχει απόλυτη επιτυχία στην επιλογή των σωστών ενεργειών. Αυτό έχει σαν αποτέλεσμα να επηρεάζει σημαντικά τον όγκο της αναζήτησης όπου είναι ο αριθμός των εφαρμόσιμων ενεργειών διακλάδωσης σε κάθε κατάσταση του χώρου καταστάσεων, οι οποίες ενέργειες πρέπει να ελεγχθούν κατά την αναζήτηση. Αν υπάρχει η υπόθεση ότι η μέση τιμή της διακλάδωσης είναι b, τότε η

πολυπλοκότητα του προβλήματος της αναζήτησης είναι της τάξεως του O(bn), όπου n είναι το μήκος της λύσης. Επίσης, η ανάστροφη αναζήτηση είναι συνήθως πιο αποτελεσματική παρά η προς τα εμπρός αναζήτηση [3].

5.4 Ευρετικά για τον Προγραμματισμό (Heuristics for Planning)

Η ευρετική συνάρτηση h(s) εκτιμά την απόσταση από την κατάσταση s στον στόχο και αυτό γίνεται εάν μπορεί να υπάρχει ένα παραδεκτό ευρετικό για την απόσταση αυτή. Μετά από αυτό καλείται ένας αλγόριθμος, για παράδειγμα ο A*, για να βρει μια βέλτιστη λύση. Επίσης, ένα αποδεκτό ευρετικό μπορεί να καθορίζεται με τον καθορισμό ενός χαλαρού προβλήματος που είναι εύκολο να λυθεί.

Ένα πρόβλημα αναζήτησης μπορεί να χαρακτηριστεί ως ένας γράφος όπου οι κόμβοι του γράφου είναι οι καταστάσεις και οι ακμές είναι οι ενέργειες. Το πρόβλημα αυτό θα καθορίσει την διαδρομή από την αρχική κατάσταση στην τελική κατάσταση. Υπάρχουν δύο τρόποι για να γίνει το πρόβλημα ποιο εύκολο. Ο πρώτος τρόπος είναι να προστεθούν περισσότερες ακμές στον γράφο, κάνοντας το έτσι αυστηρά πιο εύκολο για να μπορεί να βρει την διαδρομή από την αρχική κατάσταση στην τελική κατάσταση. Ο δεύτερος τρόπος είναι να ομαδοποιηθούν πολλαπλοί κόμβοι μαζί σχηματίζοντας έτσι ένα υποσύνολο του χώρου καταστάσεων.

Αρχικά, παρατηρούνται τα ευρετικά τα οποία προσθέτουν ακμές στον γράφο. Για παράδειγμα μπορεί να γίνει η υπόθεση ότι αγνοούνται οι προϋποθέσεις των ευρετικών φεύγοντας τις προϋποθέσεις από όλες τις ενέργειες. Κάθε ενέργεια τίθεται σε εφαρμογή και κάθε μονός στόχος μπορεί να επιτευχθεί σε ένα βήμα. Ο αριθμός των βημάτων που απαιτείται για να επιλυθεί το πρόβλημα αυτό είναι ο αριθμός των ανικανοποίητων στόχων.

Μια άλλη δυνατότητα είναι να αγνοηθεί η λίστα διαγραφής ευρετικών. Με αυτό τον τρόπο δημιουργείται μια πιο χαλαρή έκδοση του αρχικού προβλήματος που είναι πιο εύκολο για να μπορεί να λυθεί όπου το μήκος της λύσης θα είναι χρήσιμο ως ένα καλό ευρετικό. Αυτό μπορεί να γίνει μετακινώντας την λίστα διαγραφής από όλες τις ενέργειες. Έχει σαν αποτέλεσμα αυτό, να γίνει μονότονη η πρόοδος προς τον στόχο. Δηλαδή, καμία ενέργεια δεν θα μπορεί να κάνει αναίρεση της προόδου που δημιουργήθηκε από μια άλλη ενέργεια.

5.5 Σχεδιασμός Βασισμένος σε Γράφο

Τα ευρετικά πολλές φορές υποφέρουν από ανακρίβιες. Μία δομή η οποία μπορεί να δώσει καλύτερη ευρετική εκτίμηση είναι ο Σχεδιασμός Βασισμένος σε Γράφο.

Ο γράφος αποτελείται από αριθμημένα επίπεδα κόμβων. Υπάρχουν κόμβοι γεγονότων στα άρτια επίπεδα και κόμβοι ενεργειών στα περιττά επίπεδα. Επίσης, υπάρχει επαναλαμβανόμενη εναλλαγή δύο φάσεων. Οι δύο φάσεις είναι η επέκταση του γράφου και η εξαγωγή λύσης. Οι ακμές του γράφου συνδέουν τα γεγονότα ενός επιπέδου με τις ενέργειες του επόμενου επιπέδου που τα έχουν ως προϋποθέσεις και τις ενέργειες ενός επιπέδου με τα γεγονότα των λιστών προσθήκης αυτών στο επόμενο επίπεδο. Επίσης, οι ενέργειες διατήρησης συμβολίζονται με noop.

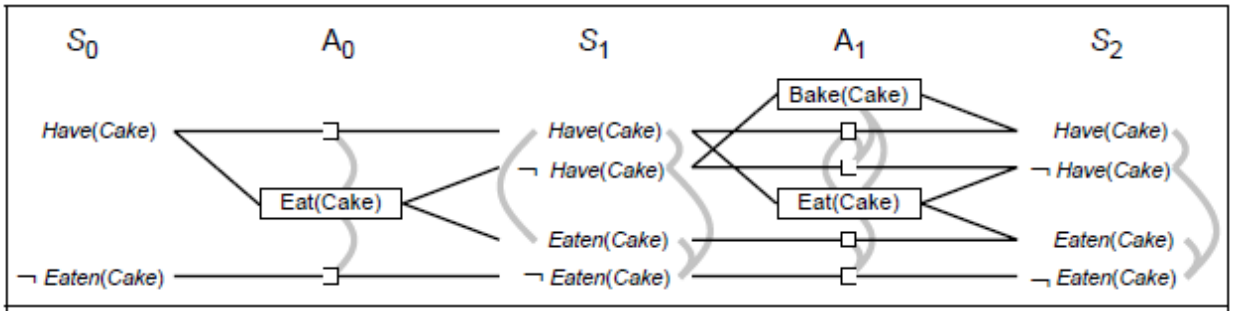
Παράδειγμα 5.2:

Init(Have(Cake))
Goal (Have (Cake) ^ Eaten (Cake))

Action (Eat(Cake)
PRECOND: Have (Cake)
EFFECT: Have (Cake) ^ Eaten (Cake))

Action (Bake Cake)
PRECOND: Have (Cake)
EFFECT: Have (Cake)

Σχήμα 5.4 Το πρόβλημα «have cake and eat cake too»



Σχήμα 5.5 Ο γράφος του προβλήματος «have cake and eat cake too»

Στο σχήμα 5.5 κάθε ενέργεια του επιπέδου A συνδέεται με τις προϋποθέσεις S_i και κάθε αποτέλεσμα με το S_{i+1}.

Μια σχέση αμοιβαίου αποκλεισμού (mutual exclusion) αναφέρεται πάντα σε δύο κόμβους του ίδιου επιπέδου και δηλώνει ότι αυτοί δεν μπορούν να βρίσκονται ταυτόχρονα στο ίδιο έγκυρο πλάνο. Δύο γεγονότα στο επίπεδο i είναι αμοιβαία αποκλειόμενα, εάν όλες οι ενέργειες στο επίπεδο i-1, συμπεριλαμβανομένων των ενεργειών noop που επιτυγχάνουν αυτά τα γεγονότα είναι μεταξύ τους αμοιβαίως αποκλειόμενες.
Η σχέση αμοιβαίου αποκλεισμού μεταξύ δύο ενεργειών που είναι στο ίδιο επίπεδο ισχύει όταν οι πιο κάτω συνθήκες ισχύουν:

- Ασυνεπή Αποτελέσματα: Μια ενέργεια αναιρεί ένα αποτέλεσμα μιας άλλης. Για παράδειγμα, Eat(cake) και η προϋπόθεση Have(cake) έχει ασυνεπή αποτελέσματα επειδή διαφωνούν με το αποτέλεσμα Have(cake).
- Παρεμβολή: Ένα από τα αποτελέσματα μιας ενέργειας αναιρεί μια προϋπόθεση της άλλης. Για παράδειγμα, Eat(cake) παρεμβαίνει με την εμμονή Have(cake) αναιρώντας την προϋπόθεση του.
- Ανταγωνιστικές Ανάγκες: Μία από τις προϋποθέσεις μιας ενέργειας είναι αμοιβαία αποκλειόμενες μαζί με τις προϋποθέσεις των άλλων. Για παράδειγμα, Bake(cake), Eat(cake) είναι mutex γιατί ανταγωνίζονται στην τιμή της προϋπόθεσης Have(cake).

Η εξαγωγή της λύσης ξεκινά σε κάποιο επίπεδο γεγονότων I μόλις εμφανιστούν όλα τα γεγονότα των στόχων, χωρίς καμιά σχέση αμοιβαίου αποκλεισμού μεταξύ τους. Τα γεγονότα των στόχων πρέπει να υποστηριχθούν από μη αμοιβαία αποκλειόμενες ενέργειες του προηγούμενου επιπέδου. Στη συνέχεια, αναδρομικά οι προϋποθέσεις των ενεργειών αυτών πρέπει να υποστηριχθούν από μη αμοιβαία αποκλειόμενες ενέργειες του προηγούμενου τους επιπέδου, μέχρι να φτάσει στο πρώτο επίπεδο. Εάν, δεν βρεθεί τέτοιο πλάνο, ο γράφος επεκτείνεται κατά 2 ακόμη επίπεδα και η διαδικασία επαναλαμβάνεται. Η συνθήκη τερματισμού είναι η εύρεση δύο εντολών που είναι στο ίδιο επίπεδο γεγονότων.

Κεφάλαιο 6

Υλοποίηση

6.1 Μέρος 1

Αρχικά, στην πρώτη φάση της υλοποίησης υλοποιήθηκε το κομμάτι στο οποίο τροποποιείται το αρχείο προβλήματος (problem). Για τη υλοποίηση του κομματιού αυτού έπρεπε να εκτελεστεί το εργαλείο LAMA. Το εργαλείο LAMA παίρνει ως είσοδο δύο αρχεία. Τα αρχεία αυτά είναι το αρχείο πεδίου ορισμού (domain) και το αρχείο προβλήματος (problem). Αφού γίνει η εκτέλεση του LAMA τότε δίνει σαν έξοδο ως αποτέλεσμα ένα αρχείο ενεργειών. Το αρχείο αυτό είναι οι ενέργειες που θα εκτελεστούν για να οδηγήσουν το πρόβλημα από την αρχική κατάσταση (initial state) στην τελική κατάσταση (goal state).

Το πρόγραμμα το οποίο υλοποιήθηκε, δέχεται σαν είσοδο τρία αρχεία. Τα αρχεία αυτά είναι το αρχείο πεδίου ορισμού (domain), το αρχείο προβλήματος (problem) και το αρχείο ενεργειών το οποίο είναι το αποτέλεσμα από το σύστημα LAMA. Στη συνέχεια, μέσα από τις διάφορες τεχνικές οι οποίες υπάρχουν στο κομμάτι αυτό γίνεται η τελική τροποποίηση του αρχείου προβλήματος (problem). Στο αρχείο αυτό κάποιες καταστάσεις έχουν διαγραφεί και

έχουν προστεθεί άλλες καταστάσεις. Αυτή η τροποποίηση γίνεται για τις μισές ενέργειες που υπάρχουν στο αρχείο ενεργειών. Πιο κάτω περιγράφεται ο τρόπος εκτέλεσης, τρόπος λειτουργίας, οι διάφορες δομές οι οποίες χρησιμοποιήθηκαν και άλλα.

6.1.1 Εντολές εκτέλεσης του προγράμματος

Για την εκτέλεση του προγράμματος αυτού είναι η εντολή: .

./main1 «domain» «problem» «sasplan»

Όπου «domain» είναι το όνομα του αρχείου πεδίου ορισμού (domain), «problem» είναι το όνομα του αρχείου προβλήματος (problem) και «sasplan» είναι το όνομα του αρχείου ενεργειών το οποίο είναι το αποτέλεσμα από το σύστημα LAMA.

Για την εκτέλεση του εργαλείου LAMA είναι η εντολή:
./plan «domain» «problem» «result»

Ή με την σειρά εντολών:

lama/translate/translate.py «domain» «problem»

lama/preprocess/preprocess < output.sas

lama/search/search fFl < output

Όπου «domain» είναι το όνομα του αρχείου πεδίου ορισμού (domain), «problem» είναι το όνομα του αρχείου προβλήματος (problem) και «result» το όνομα του αρχείου για την έξοδο του συστήματος.

6.1.2 Δομές Δεδομένων

Στο κομμάτι αυτό δημιουργήθηκαν οι πιο κάτω τύποι δεδομένων:

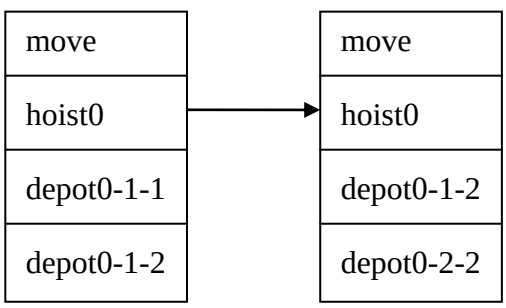
Τύπος Δεδομένων 1:

```
typedef struct METAVLITI{
struct METAVLITI *next;
char params[50000][300];
}metavliti;
```

Ο τύπος δεδομένων «metavliti» έχει δημιουργηθεί για την αποθήκευση των ενεργειών του αρχείου ενεργειών. Ουσιαστικά, είναι μία συνδεδεμένη λίστα στην οποία αποθηκεύονται οι συμβολοσειρές από το αρχείο ενεργειών. Η μεταβλητή params[50000][300] είναι ένας πίνακας που αποθηκεύονται μέσα σε αυτήν οι συμβολοσειρές. Η μεταβλητή «*next» είναι ένας δείκτης που δείχνει στο επόμενο κόμβο της λίστας. Όταν η μεταβλητή «*next» είναι ίσο με NULL τότε σημαίνει ότι η λίστα δεν έχει άλλους κόμβους, άρα είμαστε στο τέλος της λίστας. Επιπρόσθετα, χρειάζεται να γίνει αποθήκευση των ενεργειών από το αρχείο ενεργειών γιατί με βάση αυτές τις ενέργειες θα γίνει η διαγραφή ή η προσθήκη των καταστάσεων. Πιο κάτω φαίνεται ένα παράδειγμα καθώς και το σχήμα αναπαράστασης 6.1.1 για τον συγκεκριμένο τύπο δεδομένων.

Παράδειγμα 6.1.1:

(move hoist0 depot0-1-1 depot0-1-2)
(move hoist0 depot0-1-2 depot0-2-2)



Σχήμα 6.1.1 Αναπαράσταση τύπου δεδομένων «metavliti»

Τύπος Δεδομένων 2:

typedef struct ACTION{

char energia[300];
char param[50000][300];
char and1[50000][300];
char not1[50000][300];
struct ACTION *epom;
}*action*;

Ο τύπος δεδομένων «action» έχει δημιουργηθεί για την αποθήκευση των ενεργειών, των παραμέτρων, της λίστας διαγραφής και της λίστας προσθήκης. Ουσιαστικά, είναι μία συνδεδεμένη λίστα στην οποία αποθηκεύονται οι συμβολοσειρές. Όλα αυτά τα δεδομένα βρίσκονται μέσα στο αρχείο πεδίου ορισμού (domain). Επιπρόσθετα, χρειάζεται να γίνει αποθήκευση των δεδομένων αυτών από το αρχείο πεδίου ορισμού γιατί με βάση τις ενέργειες που έχουν αποθηκευτεί στον προηγούμενο τύπο δεδομένων θα γίνει αναζήτηση για να βρεθεί η συγκεκριμένη ενέργεια για να γίνει η διαγραφή των καταστάσεων που είναι αποθηκευμένες στην λίστα διαγραφής ή η προσθήκη των καταστάσεων που είναι αποθηκευμένες στην λίστα προσθήκης. Πιο κάτω είναι οι τύποι δεδομένων αυτής της δομής.

- Η μεταβλητή «energia[300]» είναι ένας πίνακας στον οποίο αποθηκεύονται συμβολοσειρές. Αυτές οι συμβολοσειρές είναι το όνομα της ενέργειας.
- Η μεταβλητή «param[50000][300]» είναι ένας πίνακας στον οποίο αποθηκεύονται συμβολοσειρές. Αυτές οι συμβολοσειρές είναι τα ονόματα των παραμέτρων.
- Η μεταβλητή «and1[50000][300]» είναι ένας πίνακας στον οποίο αποθηκεύονται συμβολοσειρές. Αυτές οι συμβολοσειρές είναι τα ονόματα των καταστάσεων που πρόκειται να προστεθούν στο αρχείο προβλήματος (problem). Επίσης, ονομάζεται σε αυτό το κομμάτι λίστα προσθήκης.
- Η μεταβλητή «not1[50000][300]» είναι ένας πίνακας στον οποίο αποθηκεύονται συμβολοσειρές. Αυτές οι συμβολοσειρές είναι τα ονόματα των καταστάσεων που πρόκειται να αφαιρεθούν από το αρχείο προβλήματος (problem). Επίσης, ονομάζεται σε αυτό το κομμάτι λίστα διαγραφής.
- Η μεταβλητή «*epom» είναι ένας δείκτης που δείχνει στο επόμενο κόμβο της λίστας.

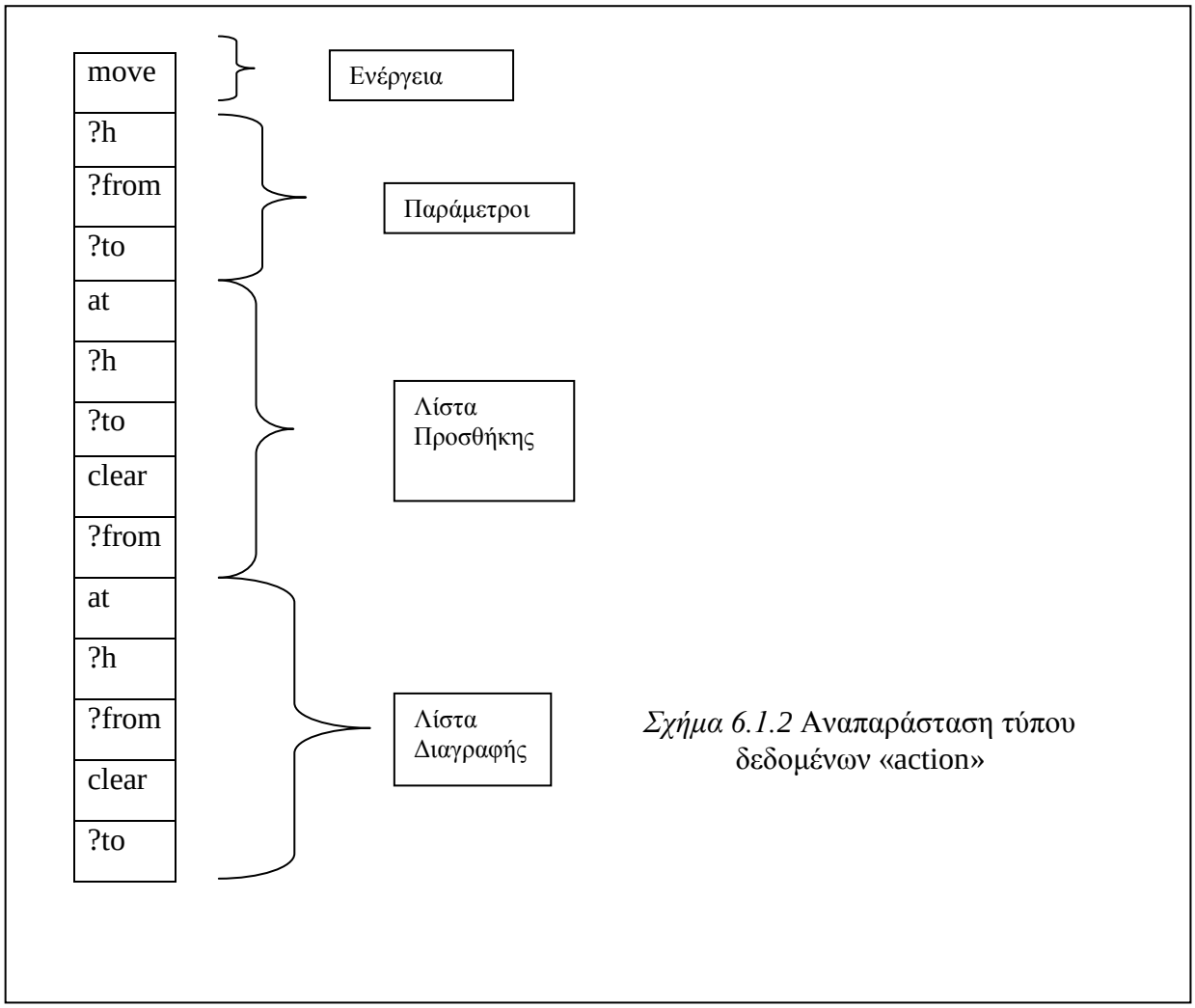
Πιο κάτω φαίνεται ένα παράδειγμα καθώς και το σχήμα αναπαράστασης 6.1.2 για τον συγκεκριμένο τύπο δεδομένων.

Παράδειγμα 6.1.2:

(:action move
 :parameters (?h - hoist ?from ?to - storearea)
 :precondition (and (at ?h ?from) (clear ?to) (connected ?from ?to))
 :effect (and (not (at ?h ?from)) (at ?h ?to) (not (clear ?to)) (clear ?from)))

Στο παραπάνω παράδειγμα η ενέργεια του αρχείου πεδίου ορισμού (domain) είναι η *move*, όπου θα αποθηκευτεί στο τύπο δεδομένων «energia». Οι παράμετροι της ενέργειας αυτής είναι οι χαρακτήρες οι οποίοι έχουν το λατινικό ερωτηματικό μπροστά τους. Δηλαδή, είναι οι μεταβλητές ?h, ?from, ?to οι οποίες θα αποθηκευτούν στην μεταβλητή «param». Επίσης, η λίστα διαγραφής για την ενέργεια αυτή είναι οι καταστάσεις (not (at ?h ?from), (not (clear

?to)) όπου δηλαδή υπάρχει μπροστά από την κατάσταση η λατινική λέξη «not». Οι ενέργειες της λίστας αυτής στη συνέχεια θα διαγραφούν από τις καταστάσεις του αρχείου προβλήματος(problem). Η λίστα προσθήκης για την συγκεκριμένη ενέργεια είναι οι καταστάσεις (at ?h ?to), (clear ?from), όπου δηλαδή πριν από τις καταστάσεις αναφέρεται η λατινική λέξη «and» ή αν δεν υπάρχει αυτή η λέξη μπροστά από τις καταστάσεις αυτές τότε πάλι είναι κατάσταση που ανήκει στην λίστα προσθήκης.



Σχήμα 6.1.2 Αναπαράσταση τύπου δεδομένων «action»

6.1.3 Τρόπος λειτουργίας του προγράμματος

Αρχικά, στο πρόγραμμα ο χρήστης δίνει την είσοδο του προγράμματος, δηλαδή τα τρία αρχεία από την γραμμή εντολών (command line). Αφού δώσει την είσοδο αυτή τότε γίνεται το άνοιγμα των αρχείων με τον κατάλληλο τύπο που πρέπει να επεξεργαστούν στο πρόγραμμα. Στη συνέχεια, με ένα μετρητή μετριούνται οι γραμμές του αρχείου ενεργειών

που έχει σαν αποτέλεσμα το σύστημα LAMA. Αυτή η διαδικασία γίνεται γιατί πρέπει να γίνει τροποποίηση για το μισό αρχείο ενεργειών.

Μετά από αυτή την διαδικασία, γίνεται επεξεργασία του αρχείου ενεργειών. Γίνεται αποθήκευση των ενεργειών στο τύπο δεδομένων «metavliti» έτσι όπως έχει παρουσιαστεί στο παράδειγμα 6.1.1 και στο σχήμα 6.1.1. Στο πρόγραμμα που έχει υλοποιηθεί διαβάζεται κάθε φορά μια συμβολοσειρά και αποθηκεύεται μέσα στο τύπο δεδομένων. Διαβάζονται συμβολοσειρές μέχρι το τέλος της γραμμής για να μπορεί να διαβαστεί η επόμενη γραμμή και να δημιουργηθεί ένας καινούργιος κόμβος για να αποθηκευτεί η νέα γραμμή στον κόμβο αυτόν. Επίσης, υπάρχει ένας μετρητής όπου αυξάνεται κάθε φορά που διαβάζεται μια καινούργια γραμμή. Η διαδικασία αυτή επαναλαμβάνεται μέχρι ο μετρητής να είναι ίσος με το μισό του μετρητή που διαβάστηκε στην αρχή του προγράμματος που μετρούσε όλες τις γραμμές του αρχείου ενεργειών.

Στη συνέχεια του προγράμματος γίνεται επεξεργασία του αρχείου πεδίου ορισμού (domain). Στο κομμάτι αυτό διαβάζεται κάθε φορά μια συμβολοσειρά. Αν αυτή η συμβολοσειρά είναι η συμβολοσειρά «action» τότε αυτό σημαίνει ότι η συμβολοσειρά που έχει διαβαστεί είναι ενέργεια και πρέπει να αποθηκευτεί η επόμενη συμβολοσειρά που θα διαβαστεί στο τύπο δεδομένων στον πίνακα ενεργειών. Αφού αποθηκευτεί η ενέργεια, τότε διαβάζεται η επόμενη συμβολοσειρά. Αν αυτή η συμβολοσειρά είναι η συμβολοσειρά «parameters» τότε πρέπει να αποθηκευτούν οι παράμετροι της ενέργειας αυτής στο τύπο δεδομένων στο πίνακα παραμέτρων. Όσες λέξεις έχουν το λατινικό ερωτηματικό μπροστά τους είναι παράμετροι. Η διαδικασία αυτή επαναλαμβάνεται για τις παραμέτρους μέχρι το τέλος της γραμμής. Στη συνέχεια αφού ολοκληρωθεί η διαδικασία αυτή διαβάζονται οι επόμενες συμβολοσειρές μέχρις ότου η συμβολοσειρά που έχει διαβαστεί είναι η συμβολοσειρά «effect». Αν είναι αυτή η συμβολοσειρά τότε σημαίνει ότι πρέπει να γίνει προσθήκη καταστάσεων στη λίστα διαγραφής και λίστα προσθήκης. Όπως έχει προαναφερθεί η λατινική συμβολοσειρά «and» σημαίνει προσθήκη καταστάσεων στη λίστα προσθήκης καταστάσεων και η λατινική συμβολοσειρά «not» σημαίνει προσθήκη καταστάσεων στη λίστα διαγραφής καταστάσεων.

Μετά από αυτή την διαδικασία αποθήκευσης των ενεργειών και των διαφόρων παραμέτρων, ακολουθεί η διαδικασία μορφοποίησης του αρχείου προβλήματος. Για να γίνει αυτή η διαδικασία μορφοποίησης πρέπει αρχικά να γίνει αναζήτηση της ενέργειας. Πρέπει να γίνει αναζήτηση μεταξύ των κόμβων του τύπου δεδομένων «metavliti» και των κόμβων «energia»

του τύπου «action». Όταν γίνει η αναζήτηση και τα δύο πεδία αυτά είναι τα ίδια τότε αρχίζει η διαδικασία δημιουργίας της λίστας προσθήκης και της λίστας διαγραφής για την συγκεκριμένη ενέργεια. Μετά από την εφαρμογή διαφόρων τεχνικών που είναι μέσω συναρτήσεων που θα επεξηγηθούν στη συνέχεια τότε προστίθενται καταστάσεις τις δύο αυτές λίστες. Στη συνέχεια, μέσω μιας συνάρτησης γίνεται η διαδικασία αντικατάστασης και διαγραφής καταστάσεων από το αρχείο προβλήματος (problem). Μετά από την τροποποίηση του αρχείου αυτού υπάρχει σαν αποτέλεσμα ένα νέο αρχείο με τις αλλαγές αυτές που έχει το ίδιο όνομα όμως με το αρχικό αρχείο προβλήματος. Η διαδικασία αυτή επαναλαμβάνεται μέχρις ότου να μην υπάρχουν άλλες ενέργειες αποθηκευμένες στους κόμβους του τύπου δεδομένων «metavliti».

Ακολούθως, πρέπει να δημιουργηθούν δύο αρχεία προβλήματος που θα είναι η έξοδος του προγράμματος. Το ένα αρχείο είναι το αρχείο που έχει σαν αποτέλεσμα η πιο πάνω διαδικασία. Το άλλο αρχείο έχει σαν αρχική κατάσταση (initial states) την αρχική κατάσταση του προβλήματος και σαν τελική κατάσταση (goal states) τις αρχικές καταστάσεις (initial states) του αρχείου που έχει σαν αποτέλεσμα η πιο πάνω διαδικασία.

Συναρτήσεις προγράμματος:

*void antigrafiarxeiou(FILE *problem, FILE *newi)*
Η συνάρτηση αυτή είναι υπεύθυνη για την αντιγραφή του αρχικού αρχείου προβλήματος σε ένα νέο αρχείο που έχει ονομαστεί newi. Η διαδικασία αυτή γίνεται γιατί πρέπει οι αρχικές καταστάσεις του αρχικού αρχείου προβλήματος πρέπει να είναι οι αρχικές καταστάσεις του νέου αρχείου και θα έχει στη συνέχεια τελική κατάσταση (goal states) τις αρχικές καταστάσεις (initial states) του αρχείου που έχει τροποποιηθεί.

*int metritis(action *n), int metritisand(action *n), int metritisparam(action *n)*
Οι συναρτήσεις αυτές έχουν υλοποιηθεί για την μέτρηση των κόμβων που έχουν αποθηκευτεί μέσα στους πίνακες «not», «and», «param» του τύπου δομών «metavliti». Η διαδικασία αυτή χρειάζεται να είναι γνωστός ο αριθμός των παραμέτρων αυτών για την μετέπειτα τους επεξεργασία καθώς επίσης να είναι γνωστός ο αριθμός παραμέτρων στη λίστα διαγραφής και λίστα προσθήκης.

*action *eisagogi(action *n,char line[])*

Η συνάρτηση αυτή έχει ως σκοπό την εισαγωγή των ενεργειών του αρχείου προβλήματος μέσα σε κόμβους. Αν η λίστα είναι κενή, δηλαδή είναι η πρώτη φορά που καλείται αυτή η συνάρτηση τότε δημιουργείται η λίστα. Αν δεν είναι κενή τότε δημιουργούνται νέοι κόμβοι και συνδέεται ο κόμβος αυτός με δείκτες πάνω στην υπόλοιπη συνδεδεμένη λίστα. Διαβάζεται κάθε φορά μια συμβολοσειρά από την κύρια συνάρτηση (main) και στέλλεται στην συνάρτηση σαν παράμετρος για την αποθήκευση τους μέσα στον πίνακα «energia» του τύπου δεδομένων «action».

*action *parametroi(action *n,char line[],int i)*

Η συνάρτηση αυτή έχει ως σκοπό την εισαγωγή των παραμέτρων του αρχείου προβλήματος μέσα σε κόμβους. Αν η λίστα είναι κενή, δηλαδή είναι η πρώτη φορά που καλείται αυτή η συνάρτηση τότε δημιουργείται η λίστα. Αν δεν είναι κενή τότε δημιουργούνται νέοι κόμβοι και συνδέεται ο κόμβος αυτός με δείκτες πάνω στην υπόλοιπη συνδεδεμένη λίστα. Διαβάζεται κάθε φορά μια συμβολοσειρά από την κύρια συνάρτηση (main) και στέλλεται στην συνάρτηση σαν παράμετρος για την αποθήκευση τους μέσα στον πίνακα «param» του τύπου δεδομένων «action».

*action *andsin(action *n,char line[],int i)*

Η συνάρτηση αυτή έχει ως σκοπό την εισαγωγή των καταστάσεων για την λίστα προσθήκης του αρχείου προβλήματος μέσα σε κόμβους. Αν η λίστα είναι κενή, δηλαδή είναι η πρώτη φορά που καλείται αυτή η συνάρτηση τότε δημιουργείται η λίστα. Αν δεν είναι κενή τότε δημιουργούνται νέοι κόμβοι και συνδέεται ο κόμβος αυτός με δείκτες πάνω στην υπόλοιπη συνδεδεμένη λίστα. Διαβάζεται κάθε φορά μια συμβολοσειρά από την κύρια συνάρτηση (main) και στέλλεται στην συνάρτηση σαν παράμετρος για την αποθήκευση τους μέσα στον πίνακα «and1» του τύπου δεδομένων «action».

*action *notsin(action *n,char line[],int i)*

Η συνάρτηση αυτή έχει ως σκοπό την εισαγωγή των καταστάσεων για την λίστα διαγραφής του αρχείου προβλήματος μέσα σε κόμβους. Αν η λίστα είναι κενή, δηλαδή είναι η πρώτη φορά που καλείται αυτή η συνάρτηση τότε δημιουργείται η λίστα. Αν δεν είναι κενή τότε δημιουργούνται νέοι κόμβοι και συνδέεται ο κόμβος αυτός με δείκτες πάνω στην υπόλοιπη συνδεδεμένη λίστα. Διαβάζεται κάθε φορά μια συμβολοσειρά από την κύρια συνάρτηση

(main) και στέλλεται στην συνάρτηση σαν παράμετρος για την αποθήκευση τους μέσα στον πίνακα «not1» του τύπου δεδομένων «action».

*metavliti *insert(metavliti *n, char line[],int i)*

Η συνάρτηση αυτή έχει ως σκοπό την εισαγωγή των ενεργειών του αρχείου ενεργειών μέσα σε κόμβους. Αν η λίστα είναι κενή, δηλαδή είναι η πρώτη φορά που καλείται αυτή η συνάρτηση τότε δημιουργείται η λίστα. Αν δεν είναι κενή τότε δημιουργούνται νέοι κόμβοι και συνδέεται ο κόμβος αυτός με δείκτες πάνω στην υπόλοιπη συνδεδεμένη λίστα. Διαβάζεται κάθε φορά μια συμβολοσειρά από την κύρια συνάρτηση (main) και στέλλεται στην συνάρτηση σαν παράμετρος για την αποθήκευση τους μέσα στον πίνακα «params» του τύπου δεδομένων «metavliti».

*void anazitisi(metavliti *n,action *a,char voith[][300],char voithand[][300],char str[])*

Η συνάρτηση αυτή αποσκοπεί στην αναζήτηση της ενέργειας που έχει αποθηκευτεί στον τύπο δεδομένων «metavliti». Γίνεται αναζήτηση αν η ενέργεια που είναι αποθηκευμένη στο τύπο δεδομένων «metavliti» που είναι οι ενέργειες στο αρχείου ενεργειών του LAMA είναι η ενέργεια που είναι αποθηκευμένη σε ένα από τους κόμβους του τύπου δεδομένων «action». Αν δεν είναι ίδιες αυτές οι δύο ενέργειες τότε ο δείκτης για την αναζήτηση ενεργειών του τύπου δεδομένων «action» πάει στον επόμενο κόμβο για να γίνει ξανά η αναζήτηση των δύο ενεργειών μέχρι να βρεθεί η ενέργεια η οποία είναι ίδια με την ενέργεια του LAMA. Αν είναι ίδιες αυτές οι δύο ενέργειες τότε αρχίζει η δημιουργία της λίστας διαγραφής. Οι παράμετροι με το λατινικό ερωτηματικό που έχουν αποθηκευτεί στον πίνακα «not1» του τύπου δεδομένων «action» αντικαθιστώνται με τις παραμέτρους της ενέργειας του αρχείου ενεργειών του LAMA. Με αυτό τον τρόπο δημιουργείται η λίστα διαγραφής.

*void anazitisiand(metavliti *n,action *a,char voithand[][300],int mparam,int numpar)*

Σε αυτή την συνάρτηση γίνεται η δημιουργία της λίστας προσθήκης της ενέργειας. Αφού οι δύο ενέργειες είναι οι ίδιες τότε οι παράμετροι με το λατινικό ερωτηματικό που έχουν αποθηκευτεί στον πίνακα «and1» του τύπου δεδομένων «action» αντικαθιστώνται με τις παραμέτρους της ενέργειας του αρχείου ενεργειών του LAMA. Με αυτό τον τρόπο δημιουργείται η λίστα προσθήκης.

*void antikatastasi(FILE*problem,FILE *out,char voith[][300],char voithand[][300],int param,int paramand,char str1[],FILE *domain,FILE *newi,int count)*

Η συνάρτηση αυτή έχει ως σκοπό την αντικατάσταση των καταστάσεων και τροποποίηση του αρχείου προβλήματος. Στην αρχή της συνάρτηση γίνεται αντιγραφή του αρχικού προβλήματος σε ένα νέο αρχείο για την μετέπειτα επεξεργασία του.

Στη συνέχεια, διαβάζεται μια συμβολοσειρά κάθε φορά από το αρχείο προβλήματος. Γίνεται μια σύγκριση μεταξύ των συμβολοσειρών που υπάρχουν στην λίστα διαγραφής και της συμβολοσειράς που διαβάστηκε. Μετά καλείται η συνάρτηση «*void elegxos(char str[],char voith[][300],int i,int *flag)*» για την σύγκριση των δύο αυτών συμβολοσειρών. Η συνάρτηση αυτή επιστρέφει τιμή 1 εάν οι δύο αυτές συμβολοσειρές είναι ίδιες. Αλλιώς, διαβάζεται άλλη συμβολοσειρά και γίνεται ξανά σύγκριση. Αν επιστραφεί η τιμή 1 από την συνάρτηση τότε καλείται η συνάρτηση «*void grammi(FILE *problem,FILE *out,int len,char voithand[][300],int param,int paramand,int mea,int nea,char str1[],int per1,int per2)*» για να γίνει η προσθήκη των καταστάσεων.

*ınt elegxosmet(FILE *domain,char voithand[][300])*

Η συνάρτηση αυτή έχει υλοποιηθεί με σκοπό την απαρίθμηση των παραμέτρων που έχει μια κατάσταση στο αρχείο πεδίου ορισμού (domain). Αυτό θα βοηθήσει μετέπειτα για την επεξεργασία του αρχείου προβλήματος και για να έχουμε γνώση πόσες παραμέτρους έχει μια κατάσταση για την αντικατάσταση τους ή την διαγραφή.

*void elegxos(char str[],char voith[][300],int i,int *flag)*

Η συνάρτηση αυτή έχει ως στόχο τον έλεγχο δύο συμβολοσειρών αν είναι ίδιες μεταξύ τους. Οι δύο αυτές συμβολοσειρές είναι η συμβολοσειρά που στέλλεται σαν παράμετρος στην συνάρτηση αυτή που έχει διαβαστεί από το αρχείο προβλήματος και η άλλη συμβολοσειρά είναι από τη λίστα διαγραφής.

*void grammi(FILE *problem,FILE *out,int len,char voithand[][300],int param,int paramand,int mea,int nea,char str1[],int per1,int per2)*

Η συνάρτηση αυτή έχει ως στόχο την αντικατάσταση του παλιού αρχείου από το νέο αρχείο προβλήματος. Διαβάζονται από το αρχείο προβλήματος οι καταστάσεις μέχρις ότου να βρεθεί η κατάσταση η οποία πρέπει να αντικατασταθεί. Υπάρχουν επίσης η περιπτώσεις στις οποίες μπορεί να πρέπει να γίνει μόνο προσθήκη καταστάσεων στο αρχείο προβλήματος. Σε αυτή την περίπτωση, στο τέλος των αρχικών καταστάσεων προστίθεται η νέα κατάσταση. Επίσης, υπάρχει η περίπτωση στην οποία μπορεί να γίνει μόνο διαγραφή μιας κατάστασης. Σε αυτή την περίπτωση διαβάζεται από το αρχείο προβλήματος οι καταστάσεις μέχρις ότου να βρεθεί η κατάσταση που πρέπει να διαγραφεί και η κατάσταση αυτή δεν αντιγράφεται στο νέο αρχείο. Μετά από αυτές τις διαδικασίες γίνεται διαγραφή του παλιού αρχείου προβλήματος και μετονομασία του νέου αρχείου με το όνομα του παλιού αρχείου.

*int euresi(action *a,char str[])*

Στην συνάρτηση αυτή γίνεται αναζήτηση της ενέργειας για να μπορούν να απαριθμηθούν οι παράμετροι που έχει η λίστα διαγραφής. Η ενέργεια αυτή που αναζητείται είναι η ενέργεια που επεξεργάζεται η συνάρτηση «anazitisi».

*int euresiand(action *a,char str[])*

Στην συνάρτηση αυτή γίνεται αναζήτηση της ενέργειας για να μπορούν να απαριθμηθούν οι παράμετροι που έχει η λίστα προσθήκης. Η ενέργεια αυτή που αναζητείται είναι η ενέργεια που επεξεργάζεται η συνάρτηση «anazitisi».

*void metatropiaxeiou(FILE *newi,FILE *problem,FILE *arx,char str1[])*

Η συνάρτηση αυτή έχει ως στόχο την δημιουργία ενός αρχείου προβλήματος το οποίο έχει σαν αρχικές καταστάσεις (initial states) το αρχείο που είχε σαν αποτέλεσμα η τροποποίηση του αρχικού αρχείου προβλήματος με την προσθήκη ή διαγραφή καταστάσεων και σαν τελικές καταστάσεις (goal states) τις τελικές καταστάσεις του αρχικού αρχείου προβλήματος.

*void metatropiaxeiouinit(FILE *newi,FILE *problem,FILE *arx,char str1[])*

Η συνάρτηση αυτή έχει ως στόχο την δημιουργία ενός αρχείου προβλήματος το οποίο έχει σαν αρχικές καταστάσεις (initial states) τις αρχικές καταστάσεις του αρχικού αρχείου προβλήματος και σαν τελικές καταστάσεις (goal states) τις τελικές καταστάσεις του αρχικές καταστάσεις (initial states) του αρχείου της προηγούμενης συνάρτησης.

Παράδειγμα εκτέλεσης 6.1.1:

```
(:action move-up-slow
  :parameters (?lift - slow-elevator ?f1 - count ?f2 - count )
  :precondition (and (lift-at ?lift ?f1) (above ?f1 ?f2 ) (reachable-floor ?lift ?f2) )
  :effect (and (lift-at ?lift ?f2) (not (lift-at ?lift ?f1)) ))

(:action move-down-slow
  :parameters (?lift - slow-elevator ?f1 - count ?f2 - count )
  :precondition (and (lift-at ?lift ?f1) (above ?f2 ?f1 ) (reachable-floor ?lift ?f2) )
  :effect (and (lift-at ?lift ?f2) (not (lift-at ?lift ?f1)) ))

(:action move-up-fast
  :parameters (?lift - fast-elevator ?f1 - count ?f2 - count )
  :precondition (and (lift-at ?lift ?f1) (above ?f1 ?f2 ) (reachable-floor ?lift ?f2) )
  :effect (and (lift-at ?lift ?f2) (not (lift-at ?lift ?f1)) ))

(:action move-down-fast
  :parameters (?lift - fast-elevator ?f1 - count ?f2 - count )
  :precondition (and (lift-at ?lift ?f1) (above ?f2 ?f1 ) (reachable-floor ?lift ?f2) )
  :effect (and (lift-at ?lift ?f2) (not (lift-at ?lift ?f1)) ))

(:action board
  :parameters (?p - passenger ?lift - elevator ?f - count ?n1 - count ?n2 - count)
  :precondition (and  (lift-at ?lift ?f) (passenger-at ?p ?f) (passengers ?lift ?n1) (next ?n1 ?n2)
(can-hold ?lift ?n2) )
  :effect (and (not (passenger-at ?p ?f)) (boarded ?p ?lift) (not (passengers ?lift ?n1))
(passengers ?lift ?n2) ))

(:action leave
  :parameters (?p - passenger ?lift - elevator ?f - count ?n1 - count ?n2 - count)
  :precondition (and  (lift-at ?lift ?f) (boarded ?p ?lift) (passengers ?lift ?n1) (next ?n2 ?n1) )
  :effect (and (passenger-at ?p ?f) (not (boarded ?p ?lift)) (not (passengers ?lift ?n1))
(passengers ?lift ?n2) ))
```

Σχήμα 6.1.3 Μορφή αρχείου πεδίου ορισμού (domain)

(lift-at fast1 n0)
(passengers fast1 n0)
(can-hold fast1 n1) (can-hold fast1 n2) (can-hold fast1 n3) (can-hold fast1 n4)
(reachable-floor fast1 n0)(reachable-floor fast1 n4)(reachable-floor fast1 n8)(reachable-floor fast1 n12)(reachable-floor fast1 n16)

(lift-at slow0-0 n0)
(passengers slow0-0 n0)
(can-hold slow0-0 n1) (can-hold slow0-0 n2) (can-hold slow0-0 n3)
(reachable-floor slow0-0 n0)(reachable-floor slow0-0 n1)(reachable-floor slow0-0 n2)(reachable-floor slow0-0 n3)(reachable-floor slow0-0 n4)(reachable-floor slow0-0 n5)(reachable-floor slow0-0 n6)(reachable-floor slow0-0 n7)(reachable-floor slow0-0 n8)

(lift-at slow1-0 n12)
(passengers slow1-0 n0)
(can-hold slow1-0 n1) (can-hold slow1-0 n2) (can-hold slow1-0 n3)
(reachable-floor slow1-0 n8)(reachable-floor slow1-0 n9)(reachable-floor slow1-0 n10)(reachable-floor slow1-0 n11)(reachable-floor slow1-0 n12)(reachable-floor slow1-0 n13)(reachable-floor slow1-0 n14)(reachable-floor slow1-0 n15)(reachable-floor slow1-0 n16)

(passenger-at p0 n13)
(passenger-at p1 n2)
(passenger-at p2 n9)
(passenger-at p3 n6)
(passenger-at p4 n9)
(passenger-at p5 n14)
(passenger-at p6 n3)
(passenger-at p7 n13)
(passenger-at p8 n14)
(passenger-at p9 n5)

Σχήμα 6.1.4 Αρχική Μορφή αρχείου προβλήματος

```
(move-up-slow slow1-0 n12 n16)
(move-down-slow slow1-0 n16 n13)
(board p7 slow1-0 n13 n0 n1)
(board p0 slow1-0 n13 n1 n2)
(move-up-slow slow1-0 n13 n16)
(move-up-slow slow0-0 n0 n6)
(board p3 slow0-0 n6 n0 n1)
(move-down-slow slow0-0 n6 n5)
(board p9 slow0-0 n5 n1 n2)
(move-up-slow slow0-0 n5 n8)
(leave p7 slow1-0 n16 n2 n1)
(leave p9 slow0-0 n8 n2 n1)
(move-down-slow slow0-0 n8 n3)
(board p6 slow0-0 n3 n1 n2)
(move-down-slow slow0-0 n3 n2)
(board p1 slow0-0 n2 n2 n3)
(move-up-slow slow0-0 n2 n8)
```

Σχήμα 6.1.5 Μορφή αρχείου ενεργειών

```
energia : move-up-slow
not: lift-at ?lift ?f2
and: lift-at ?lift ?f1

energia : move-down-slow
not: lift-at ?lift ?f2
and: lift-at ?lift ?f1

energia : move-up-fast
not: lift-at ?lift ?f2
3. lift-at ?lift ?f1

energia : move-down-fast
not: lift-at ?lift ?f2
and: lift-at ?lift ?f1

energia : board
not: boarded ?p ?lift passengers ?lift ?n2
and: passenger-at ?p ?f passengers ?lift ?n1

energia : leave
not: passenger-at ?p ?f passengers ?lift ?n2
and:  boarded ?p ?lift passengers ?lift ?n1
```

Σχήμα 6.1.6 Αναπαράσταση της αποθήκευσης μέσα στους κόμβους για τον τύπο δεδομένων «action» και για τους πίνακες αυτού του τύπου «energia», «not1», «and1»

(lift-at fast1 n0)
(passengers fast1 n0)
(can-hold fast1 n1) (can-hold fast1 n2) (can-hold fast1 n3) (can-hold fast1 n4)
(reachable-floor fast1 n0)(reachable-floor fast1 n4)(reachable-floor fast1 n8)(reachable-floor fast1 n12)(reachable-floor fast1 n16)

(lift-at slow0-0 n1)
(passengers slow0-0 n1)
(can-hold slow0-0 n1) (can-hold slow0-0 n2) (can-hold slow0-0 n3)
(reachable-floor slow0-0 n0)(reachable-floor slow0-0 n1)(reachable-floor slow0-0 n2)(reachable-floor slow0-0 n3)(reachable-floor slow0-0 n4)(reachable-floor slow0-0 n5)(reachable-floor slow0-0 n6)(reachable-floor slow0-0 n7)(reachable-floor slow0-0 n8)

(lift-at slow1-0 n8)
(passengers slow1-0 n1)
(can-hold slow1-0 n1) (can-hold slow1-0 n2) (can-hold slow1-0 n3)
(reachable-floor slow1-0 n8) (reachable-floor slow1-0 n9) (reachable-floor slow1-0 n10)(reachable-floor slow1-0 n11)(reachable-floor slow1-0 n12)(reachable-floor slow1-0 n13)(reachable-floor slow1-0 n14)(reachable-floor slow1-0 n15)(reachable-floor slow1-0 n16)

(boarded p0 slow1-0)
(passenger-at p1 n8)
(passenger-at p2 n8)
(boarded p3 slow0-0)
(passenger-at p4 n16)
(passenger-at p5 n14)
(passenger-at p6 n8)
(passenger-at p7 n16)
(passenger-at p8 n14)
(passenger-at p9 n8)

Σχήμα 6.1.7 Μορφή του τελικού αρχείου προβλήματος

Στο παράδειγμα 6.1.1 είναι ένα παράδειγμα εκτέλεσης του πρώτου Μέρους της υλοποίησης. Το αρχικό πεδίο ορισμού είναι το Σχήμα 6.1.3, το αρχικό πρόβλημα παρουσιάζεται στο Σχήμα 6.1.4 και το αρχείο ενεργειών παρουσιάζεται στο αρχείο 6.1.5. Με τις διάφορες τεχνικές που έχουν περιγραφεί πιο πάνω προκύπτει το τελικό αρχείο προβλήματος που παρουσιάζεται στο Σχήμα 6.1.7. Παρατηρούνται αλλαγές στις καταστάσεις από το αρχικό στο τελικό αρχείο προβλήματος.

6.2 Μέρος 2

Ο σκοπός του κομματιού αυτού ήταν η συμπίεση του μήκους του αρχείου ενεργειών που είχε σαν αποτέλεσμα το σύστημα LAMA. Δηλαδή, να μειωθεί ο συνολικός αριθμός των ενεργειών που υπάρχουν σε ένα αρχείο ενεργειών. Μέσα από διάφορες τεχνικές που θα περιγραφούν παρακάτω ο στόχος αυτός έχει επιτευχθεί.

Το πρόγραμμα αυτό δέχεται σαν είσοδο ένα αρχείο ενεργειών. Στην υλοποίηση του μέρους αυτού επιλέγεται το υποσύνολο των ενεργειών για τις γραμμές 25%-75% του αρχείου ενεργειών. Μετά καλείται το πρόγραμμα που υλοποιήθηκε στο Μέρος 1 για την τροποποίηση του αρχείου προβλήματος με αυτό το υποσύνολο για αρχείο ενεργειών. Σκοπός της επιλογής του υποσυνόλου είναι να γίνει πιο μικρό το μήκος του υποσυνόλου αυτού.

Στη συνέχεια, καλείται το εργαλείο SATPLAN. Το SATPLAN δέχεται σαν είσοδο δύο αρχεία. Τα αρχεία αυτά είναι το αρχείο πεδίο ορισμού και το αρχείο προβλήματος. Μετά από την εκτέλεση του SATPLAN έχει σαν έξοδο ένα αρχείο. Ο SATPLAN με κωδικοποιητή SAT-MAX-PLAN (SMP) έχει ως στόχο να βρίσκει λύσεις με το μικρότερο μήκος.

Μετέπειτα, το υποσύνολο το οποίο επιθυμείται να γίνει πιο μικρό το μήκος του αντικαθιστάται από το αρχείο που έχει σαν έξοδο το πρόγραμμα SATPLAN. Γίνεται η ίδια διαδικασία για το υποσύνολο των γραμμών του αρχείου ενεργειών 0%-50%.

6.2.1 Τρόπος λειτουργίας

Αρχικά, στο πρόγραμμα ο χρήστης δίνει την είσοδο του προγράμματος, δηλαδή το αρχείο ενεργειών από την γραμμή εντολών (command line). Αφού δώσει την είσοδο αυτή τότε γίνεται το άνοιγμα του αρχείου με τον κατάλληλο τύπο που πρέπει να επεξεργαστεί στο πρόγραμμα. Στη συνέχεια, το υποσυνόλου του αρχείου ενεργειών γίνεται το ποσοστό 25% με 75% των γραμμών του αρχείου αυτού. Μετά, αριθμείται ο αριθμός των γραμμών που υπάρχουν μέσα στο αρχείο.

Μετέπειτα, καλείται το πρόγραμμα που έχει υλοποιηθεί στο πρώτο Μέρος. για το υποσύνολο 25%-75% το πρόγραμμα θα πρέπει να εκτελεστεί δύο φορές. Μία φορά από το 0%-25% και μια δεύτερη φορά από το 0%-75%. Το τελικό αρχείο προβλήματος θα έχει αρχικές καταστάσεις (initial states) τις καταστάσεις που είχε σαν αποτέλεσμα όταν εκτελέστηκε το

πρόγραμμα από 0%-25% και σαν τελικές καταστάσεις (goal states) τις καταστάσεις που είχε σαν αποτέλεσμα όταν εκτελέστηκε το πρόγραμμα από 0%-75%. Στη συνέχεια, καλείται για εκτέλεση ο SATPLAN. Επιστρέφει ένα αρχείο ως έξοδο. Στη συνέχεια, αντικαθιστάται το κομμάτι αυτό στο υποσύνολο που έχει δώσει ο χρήστης.

Συναρτήσεις προγράμματος:

*void xorismos_arxeion(FILE *out,FILE *sas,int grammes,int *a,int *b)*

Η συνάρτηση αυτή έχει ως στόχο τον χωρισμό του αρχείου σε μικρότερα υποσύνολα.

*void prosarmogi_arxeiou(FILE *in,FILE *sas,FILE *new,int min,int max,char *arg,int num)*

Σε αυτή την συνάρτηση γίνεται αντικατάσταση του κομματιού που επιλέχτηκε για επεξεργασία με το κομμάτι που είχε σαν αποτέλεσμα ο SATPLAN.

*int metrimagrammon(FILE *sas)*

Η συνάρτηση αυτή αποσκοπεί στο να απαριθμήσει τις γραμμές του αρχείου ενεργειών έτσι ώστε να βοηθήσει μετέπειτα στο χώρισμα του αρχείου με βάση τα ποσοστά υποσυνόλου.

Παράδειγμα 6.2.1:

```
(move-down-fast fast1 n6 n2)
(move-down-fast fast1 n2 n0)
(board p3 fast1 n0 n0 n1)
(move-up-fast fast1 n0 n2)
(leave p3 fast1 n2 n1 n0)
(move-up-slow slow1-0 n4 n5)
(move-down-slow slow0-0 n4 n3)
(board p0 slow0-0 n3 n0 n1)
(move-down-slow slow0-0 n3 n1)
(board p1 slow0-0 n1 n1 n2)
(move-up-slow slow0-0 n1 n4)
(leave p1 slow0-0 n4 n2 n1)
(leave p0 slow0-0 n4 n1 n0)
(move-up-slow slow1-0 n5 n7)
(board p2 slow1-0 n7 n0 n1)
(move-down-slow slow1-0 n7 n5)
(move-down-slow slow1-0 n5 n4)
(board p1 slow1-0 n4 n1 n2)
(move-up-slow slow1-0 n4 n5)
(leave p1 slow1-0 n5 n2 n1)
(move-up-slow slow1-0 n5 n6)
(leave p2 slow1-0 n6 n1 n0)
```

Σχήμα 6.2.1 Το αρχείο ενεργειών

```
; Time 0.05
; ParsingTime    0.00
; MakeSpan 3
0: (BOARD P3 FAST1 N0 N0 N1) [1]
0: (MOVE-DOWN-SLOW SLOW0-0 N4 N3) [1]
0: (MOVE-UP-SLOW SLOW1-0 N4 N8) [1]
1: (BOARD P0 SLOW0-0 N3 N0 N1) [1]
1: (MOVE-UP-FAST FAST1 N0 N2) [1]
2: (LEAVE P3 FAST1 N2 N1 N0) [1]
2: (MOVE-DOWN-SLOW SLOW1-0 N8 N5) [1]
```

Σχήμα 6.2.2 Αποτέλεσμα του συστήματος SATPLAN

(move-down-fast fast1 n6 n2)
(move-down-fast fast1 n2 n0)
(board p3 fast1 n0 n0 n1)
(move-up-fast fast1 n0 n2)
0: (BOARD P3 FAST1 N0 N0 N1) [1]
0: (MOVE-DOWN-SLOW SLOW0-0 N4 N3) [1]
0: (MOVE-UP-SLOW SLOW1-0 N4 N8) [1]
1: (BOARD P0 SLOW0-0 N3 N0 N1) [1]
1: (MOVE-UP-FAST FAST1 N0 N2) [1]
2: (LEAVE P3 FAST1 N2 N1 N0) [1]
2: (MOVE-DOWN-SLOW SLOW1-0 N8 N5) [1]
(move-down-slow slow1-0 n5 n4)
(board p1 slow1-0 n4 n1 n2)
(move-up-slow slow1-0 n4 n5)
(leave p1 slow1-0 n5 n2 n1)
(move-up-slow slow1-0 n5 n6)
(leave p2 slow1-0 n6 n1 n0)

Σχήμα 6.2.3 Το τελικό αρχείο ενεργειών

Στο παράδειγμα 6.2.1 είναι ένα παράδειγμα εκτέλεσης του δεύτερου Μέρους της υλοποίησης. Το πεδίο ενεργειών είναι το Σχήμα 6.2.1, το αποτέλεσμα του συστήματος SATPLAN παρουσιάζεται στο Σχήμα 6.2.2 και το τελικό αρχείο ενεργειών παρουσιάζεται στο αρχείο 6.2.3. Με τις διάφορες τεχνικές που έχουν περιγραφεί πιο πάνω προκύπτει το τελικό αρχείο προβλήματος που παρουσιάζεται στο Σχήμα 6.2.3. Παρατηρείται αλλαγές τόσο στο μήκους του αρχείου ενεργειών όσο και στις καταστάσεις στις οποίες μπορούν να εκτελεστούν ταυτόχρονα. Για παράδειγμα οι καταστάσεις, που αριθμούνται με τον ίδιο αριθμό τότε αυτές μπορούν να εκτελεστούν ταυτόχρονα.

6.3 Μέρος 3

Στο μέρος αυτό έγινε η ένωση σε ένα πρόγραμμα των δύο πιο πάνω προγραμμάτων, δηλαδή του Μέρος 1 και του Μέρος 2. Η υλοποίηση αυτή είχε ως στόχο να γίνει συμπίεση του αρχείου των ενεργειών που είχε σαν έξοδο το πρόγραμμα LAMA. Με τον όρο συμπίεση σημαίνει να μειωθεί ο συνολικός αριθμός των ενεργειών που υπάρχουν στο αρχείο ενεργειών.

6.3.1 Τρόπος λειτουργίας

Αρχικά, στο πρόγραμμα ο χρήστης δίνει την είσοδο του προγράμματος, δηλαδή το αρχείο πεδίου ορισμού (domain), αρχείο προβλήματος (problem), και το αρχείο ενεργειών που είναι αποτέλεσμα από το σύστημα LAMA από την γραμμή εντολών (command line). Αφού δώσει την είσοδο αυτή τότε γίνεται το άνοιγμα των αρχείων με τον κατάλληλο τύπο για κάθε αρχείο. Στη συνέχεια, γίνεται διαμοιρασμός του αρχείου ενεργειών για το υποσύνολο των γραμμών 25%-75% για να γίνει επεξεργασία για την σμίκρυνση του αρχείου. Αφού γίνει αυτός ο διαμοιρασμός του αρχείου τότε καλείται η συνάρτηση όπου γίνεται η τροποποίηση του αρχείου με την πρόσθεση και διαγραφή των καταστάσεων έτσι όπως έχει περιγραφεί στο Μέρος 1. Η διαδικασία αυτή γίνεται για δύο αρχεία ενεργειών. Για το αρχείο από 0%-25% των γραμμών του αρχείου ενεργειών και για το αρχείο 0%-75% των γραμμών του αρχείου ενεργειών. Αφού γίνει εκτέλεση για τα δύο αρχεία αυτά τότε το τελικό αρχείο προβλήματος είναι να έχει αρχικές καταστάσεις (initial states) τις καταστάσεις που έχει σαν αποτέλεσμα η εκτέλεση του προγράμματος για το πρώτο αρχείο και να έχει τελικές καταστάσεις (goal states) τις καταστάσεις που έχει σαν αποτέλεσμα η εκτέλεση του προγράμματος για το δεύτερο αρχείο. Μετά από αυτή την διαδικασία εκτελείται το σύστημα SATPLAN. Το αποτέλεσμα του συστήματος SATPLAN, που είναι ένα αρχείο ενεργειών στο οποίο κάποιες ενέργειες μπορούν να εκτελούνται παράλληλα. Το αρχείο αυτό ενσωματώνεται στο αρχικό αρχείο ενεργειών στο κομμάτι των γραμμών 25%-75%.

Μετά από το υποσύνολο 25%-75%, γίνεται η επεξεργασία του νέου αρχείου για το κομμάτι των γραμμών του αρχείου 0%-50%. Εκτελείται η συνάρτηση του Μέρους 1 όμως τώρα για αρχείο ενεργειών το αρχείο για τις γραμμές από 0% μέχρι 50%. Μετά από αυτή την διαδικασία εκτελείται το σύστημα SATPLAN. Ο SATPLAN θα εκτελεστεί δύο φορές. Μία φορά που θα έχει αρχικές καταστάσεις (initial states) τις αρχικές καταστάσεις του αρχικού

προβλήματος και τελικές καταστάσεις τις καταστάσεις που είχε σαν αποτέλεσμα η εκτέλεση του Μέρους 1. Ακόμη μία φορά, όπου θα έχει αρχικές καταστάσεις (initial states) τις αρχικές καταστάσεις που είχε σαν αποτέλεσμα η εκτέλεση του Μέρους 1 και τελικές καταστάσεις του αρχικού προβλήματος. Το πρώτο αποτέλεσμα του συστήματος SATPLAN, ενσωματώνεται στο αρχείο ενεργειών στο κομμάτι των γραμμών 0%-50% και το δεύτερο αποτέλεσμα του συστήματος SATPLAN ενσωματώνεται στο αρχείο ενεργειών στο κομμάτι των γραμμών 50%-100%. Στο τέλος της εκτέλεσης του προγράμματος υπολογίζεται και ο χρόνος που χρειάζεται για να εκτελεστεί το πρόγραμμα. Ο χρόνος υπολογίζεται σε δευτερόλεπτα (sec).

Όταν τελειώσει η πιο πάνω διαδικασία εκτελείται το πρόγραμμα «validate» έτσι ώστε να αξιολογηθούν τα αρχεία των ενεργειών αν ικανοποιούνται οι αρχικές και οι τελικές καταστάσεις του προβλήματος. Αν ικανοποιούνται τότε το πρόγραμμα αυτό βγάζει «success» που σημαίνει ότι το αρχείο ενεργειών είναι σωστό αλλιώς βγάζει «failed» που σημαίνει ότι το αρχείο ενεργειών είναι λάθος και δεν ικανοποιούνται κάποιες καταστάσεις.

6.3.2 Εγχειρίδιο χρήσης του προγράμματος

Το πρόγραμμα που έχει περιγραφεί πιο πάνω δέχεται σαν είσοδο από την γραμμή εντολών (command line) τρία αρχεία. Λόγω του ότι το κάθε αρχείο έχει διαφορετική δομή που είναι αποθηκευμένα τότε αποφασίστηκε να ακολουθηθεί μία συγκεκριμένη δομή για όλα τα αρχεία. Για την σωστή λειτουργία του προγράμματος τα αρχεία αν δεν ακολουθούν την συγκεκριμένη δομή το πρόγραμμα δεν θα λειτουργεί σωστά για αυτό τον λόγο θα πρέπει να προσαρμόζονται.

Για το αρχείο πεδίου ορισμού (domain) για την λέξη «:effect» θα πρέπει να ισχύει η μορφή που φαίνεται στο πιο κάτω σχήμα 6.3.1. Δηλαδή, θα πρέπει όλες οι καταστάσεις να είναι σε μία γραμμή και διαχωριζόμενες μεταξύ τους με ένα κενό διάστημα. Επίσης, στο τέλος των καταστάσεων υπάρχει το κλείσιμο των παρενθέσεων πρέπει να υπάρχει ένα κενό διάστημα. Πρέπει η προτελευταία κατάσταση να είναι η παρένθεση του κλεισίματος που έχει με την λέξη "effect" και η τελευταία παρένθεση να είναι το κλείσιμο της παρένθεσης που βρίσκεται στην αρχή της λέξης "action". Ακόμη, όπου αναγράφεται η λέξη «(:predicates» δεν θα πρέπει να υπάρχουν παρενθέσεις εσωτερικά των γραμμάτων. Στο σχήμα δείγματος 6.3.1 για το πώς πρέπει να είναι το αρχείο πεδίου ορισμού.

```
(:predicates
        (passenger-at ?person - passenger ?floor - count)
        (boarded ?person - passenger ?lift - elevator)
        (lift-at ?lift - elevator ?floor - count)
        (reachable-floor ?lift - elevator ?floor - count)
        (above ?floor1 - count ?floor2 - count)
        (passengers ?lift - elevator ?n - count)
        (can-hold ?lift - elevator ?n - count)
        (next ?n1 - count ?n2 - count)
)

(:action move-up-slow
  :parameters (?lift - slow-elevator ?f1 - count ?f2 - count )
  :precondition (and (lift-at ?lift ?f1) (above ?f1 ?f2 ) (reachable-floor ?lift ?f2) )
  :effect (and (lift-at ?lift ?f2) (not (lift-at ?lift ?f1)) ))
```

Σχήμα 6.3.1 Αρχείο Πεδίου Ορισμού

Για το αρχείο προβλήματος (problem) θα πρέπει όπου γράφεται η λέξη «(:init» θα πρέπει να είναι σε νέα γραμμή. Επίσης, οι λέξεις «(:goal» και «(and» θα πρέπει να είναι σε νέα γραμμή. Ακόμη, στο τέλος των «(:goal» θα πρέπει να έχει δύο παρενθέσεις «))» οι οποίες πρέπει να είναι σε νέα γραμμή. Επίσης, στο τέλος να υπάρχει ακόμη μία παρένθεση «)». Στο σχήμα δείγματος 6.3.2 για το πώς πρέπει να είναι το αρχείο προβλήματος.

```
(:init
(next n0 n1) (next n1 n2) (next n2 n3) (next n3 n4) (next n4 n5) (next n5 n6) (next n6 n7)
(next n7 n8) (next n8 n9) (next n9 n10) (next n10 n11) (next n11 n12) (next n12 n13) (next
n13 n14) (next n14 n15) (next n15 n16)
(above n0 n1) (above n0 n2) (above n0 n3) (above n0 n4) (above n0 n5) (above n0 n6)
(above n0 n7) (above n0 n8) (above n0 n9) (above n0 n10) (above n0 n11) (above n0 n12)
(above n0 n13) (above n0 n14) (above n0 n15) (above n0 n16)
(above n1 n2) (above n1 n3) (above n1 n4) (above n1 n5) (above n1 n6) (above n1 n7)
)

(:goal
(and
(passenger-at p0 n15)
 (passenger-at p8 n10)
(passenger-at p9 n16)
))
)
```

Σχήμα 6.3.2 Αρχείο Προβλήματος

6.3.3 Εντολές εκτέλεσης του προγράμματος

Για την εκτέλεση του προγράμματος αυτού είναι η εντολή: .

./main «domain» «problem» «sasplan»

Όπου «domain» είναι το όνομα του αρχείου πεδίου ορισμού (domain), «problem» είναι το όνομα του αρχείου προβλήματος (problem) και «sasplan» είναι το όνομα του αρχείου ενεργειών το οποίο είναι το αποτέλεσμα από το σύστημα LAMA.

Για την εκτέλεση του συστήματος LAMA είναι η εντολή:
./plan «domain» «problem» «result»

Ή με την σειρά εντολών:

lama/translate/translate.py «domain» «problem»

lama/preprocess/preprocess < output.sas

lama/search/search fFl < output

Όπου «domain» είναι το όνομα του αρχείου πεδίου ορισμού (domain), «problem» είναι το όνομα του αρχείου προβλήματος (problem) και «result» το όνομα του αρχείου για την έξοδο του συστήματος.

Κεφάλαιο 7

Πειραματική Ανάλυση

Στο κεφάλαιο αυτό περιγράφονται οι πειραματικές αξιολογήσεις που έχουν γίνει κατά την διάρκεια της παρούσας διπλωματικής εργασίας. Έχουν γίνει πειραματικές μετρήσεις για τον χρόνο εκτέλεσης και για τα επίπεδα επίλυσης ενός προβλήματος του LAMA, SATPLAN καθώς επίσης και του ενοποιημένου συστήματος των δύο προαναφερθέντων συστημάτων.

7.1 Μέτρηση χρόνου εκτέλεσης του SATPLAN

Για την επίλυση των προβλημάτων προγραμματισμού δράσης χρησιμοποιείται το εργαλείο SATPLAN για την προσαρμογή σε προβλήματα ικανοποιησιμότητας προτασιακής λογικής. Έτσι όπως περιγράφεται στο Κεφάλαιο 4 η λειτουργία του εργαλείου αυτού έχει ως στόχο να βρίσκει λύσεις με το μικρότερο μήκος. Το εργαλείο αυτό κατασκευάζει αρχικά ένα γράφο μέχρι ένα επίπεδο και μετά μεταφράζει τους περιορισμούς οι οποίοι προκύπτουν σε Συζευκτική Κανονική Μορφή (CNF) για να μπορεί να δώσει τους περιορισμούς αυτούς σε ένα προτασιακό επιλυτή. Σε αυτή την Διπλωματική Εργασία, χρησιμοποιείται ο κωδικοποιητής SAT-MAX-PLAN (SMP) .

Αρχικά σε αυτή την ενότητα το εργαλείο SATPLAN δεχόταν σαν είσοδο ένα αρχείο πεδίου ορισμού (domain) και ένα αρχείο προβλήματος (problem). Στη συνέχεια, το εργαλείο SATPLAN έτρεχε με βάση τα αρχεία που δίνονταν στην είσοδο. Όταν τελείωνε η εκτέλεση γινόταν καταγραφή του χρόνου ολοκλήρωσης του εργαλείου (real time, system time) καθώς επίσης και των επιπέδων (levels) που χρειάστηκε το σύστημα για να επιλύσει το πρόβλημα που δεχόταν στην είσοδο.

Οι μετρήσεις για αυτή την πειραματική αξιολόγηση έγιναν με βάση τα προβλήματα Elevator, Transport. Επίσης, στις μετρήσεις δόθηκε χρονικό περιθώριο (timeout) για την ολοκλήρωση του SATPLAN μία ώρα. Δηλαδή, αν το εργαλείο δεν επιλύσει το πρόβλημα μέσα στο περιθώριο της μιας ώρας τότε θεωρείται ότι το πρόβλημα δεν επιλύεται. Αυτή η περίπτωση συμβολίζεται μέσα στους πίνακες με τον χαρακτήρα (-). Τα αποτελέσματα περιγράφονται στους πίνακες 7.1.1, 7.1.2.

Μέτρηση χρόνου και επιπέδων για το σύστημα SATPLAN **Domain: Elevator**			
Domain-Problem	Επίπεδα (levels)	Πραγματικός Χρόνος(real time) (sec)	Χρόνος Συστήματος (System time) (sec)
p01	8	0.49	0.19
p02	11	2.51	0.04
p03	7	1.32	0.07
p04	11	4.37	0.24
p05	9	3.73	0.18
p06	14	20.86	0.7
p07	15	39.57	1.02
p08	17	63.62	1.72
p09	13	38.19	1.09
p10	22	808.23	6.52
p11	21	896.48	6.78
p12	25	-	-
p13	20	508.91	7.37

p14	33	-	-
p15	38	-	-
p16	28	_	_
p17	35	_	_
p18	46	-	-
p19	-	_	_
p20	43	-	-

Πίνακας 7.1.1 Πρόβλημα Elevator

Μέτρηση χρόνου και επιπέδων για το σύστημα SATPLAN Domain: Transport			
Domain-Problem	Επίπεδα (levels)	Πραγματικός Χρόνος(real time) (sec)	Χρόνος Συστήματος (System time) (sec)
p01	4	0.31	0.05
p02	11	14.47	1.5
p03	11	140.49	15.69
p04	14	1241.22	81.7
p05	25	-	-
p06	27	-	-
p07	28	-	-
p08	-	-	-
p09	-	-	-
p10	-	-	-
p11	8	-	-
p12	18	-	-
p13	33	-	-
p14	27	-	-
p15	35	-	-
p16	39	-	-
p17	-	-	-

p18	-	-	-
p19	-	-	-
p20	-	-	-

Πίνακας 7.1.2 Πρόβλημα Transport

7.2 Μέτρηση χρόνου εκτέλεσης του LAMA

Ένας άλλος τρόπος προσέγγισης για την επίλυση των προβλημάτων Προγραμματισμού Δράσης είναι η ευρετική αναζήτηση όπου χρησιμοποιείται το εργαλείο LAMA. Το LAMA όπως έχει αναπτυχθεί η λειτουργία του στο Κεφάλαιο 4, υπολογίζει στην αρχή μια γρήγορη λύση με τη χρήση ενός αποδοτικού αλγορίθμου και μετά γίνεται αναζήτηση για μικρότερου μήκους πλάνα καλώντας διαδοχικές εκτελέσεις του A* αλγορίθμου με μειωμένα τα βάρη.

Αρχικά σε αυτή την ενότητα το LAMA δεχόταν σαν είσοδο ένα αρχείο πεδίου ορισμού (domain) και ένα αρχείο προβλήματος (problem). Στη συνέχεια, το LAMA έτρεχε με βάση τα αρχεία που δίνονταν στην είσοδο. Όταν τελείωνε η εκτέλεση γινόταν καταγραφή του χρόνου αναζήτησης, ολικού χρόνου καθώς επίσης και των επιπέδων (levels) που χρειάστηκε το εργαλείο για να επιλύσει το πρόβλημα που δεχόταν στην είσοδο.

Οι μετρήσεις για αυτή την πειραματική αξιολόγηση έγιναν με βάση τα προβλήματα Elevator και Transport. Επίσης, στις μετρήσεις δόθηκε χρονικό περιθώριο (timeout) για την ολοκλήρωση του LAMA μία ώρα. Δηλαδή, αν το εργαλείο δεν επιλύσει το πρόβλημα μέσα στο περιθώριο της μιας ώρας τότε θεωρείται ότι το πρόβλημα δεν επιλύεται. Αυτή η περίπτωση συμβολίζεται μέσα στους πίνακες με τον χαρακτήρα (-). Τα αποτελέσματα περιγράφονται στους πίνακες 7.2.1, 7.2.2.

Μέτρηση χρόνου και επιπέδων για το σύστημα LAMA **Domain: Elevator**			
Domain-Problem	Επίπεδα (levels)	Χρόνος Αναζήτησης (Search Time) (sec)	Ολικός Χρόνος (Total time) (sec)
p01	22	0	0.01
p02	24	0	0.02
p03	24	0	0.04
p04	36	0.01	0.04

p05	32	0.01	0.05
p06	48	0.06	0.1
p07	65	0.04	0.08
p08	48	0.03	0.09
p09	51	0.02	0.08
p10	77	0.16	0.23
p11	34	0.03	0.11
p12	54	0.09	0.2
p13	59	0.16	0.28
p14	82	0.36	0.52
p15	86	0.44	0.62
p16	80	0.48	0.69
p17	98	1.1	1.36
p18	102	0.55	0.83
p19	128	1.77	2.12
p20	106	0.9	1.22

Πίνακας 7.2.1 Πρόβλημα Elevator

Μέτρηση χρόνου και επιπέδων για το σύστημα LAMA **Domain: Transport**			
Domain-Problem	Επίπεδα (levels)	Χρόνος Αναζήτησης (Search Time) (sec)	Ολικός Χρόνος (Total time) (sec)
p01	6	0	0
p02	21	0	0.03
p03	36	0.05	0.15
p04	57	0.19	0.39
p05	72	0.61	0.98
p06	88	2.59	3.45
p07	107	7.42	8.6
p08	103	9.02	10.54
p09	125	17.66	19.82
p10	140	27.6	30.29
p11	11	0	0
p12	30	0.01	0.04
p13	41	0.02	0.03
p14	62	0.2	0.41
p15	110	1.72	1.87
p16	142	17.67	18.39
p17	218	38.06	39.35

p18	232	150.24	151.93
p19	340	553.27	555.77
p20	327	661.35	664.78

Πίνακας 7.2.2 Πρόβλημα Transport

7.3 Μέτρηση χρόνου με συνδυασμό προσεγγίσεων επίλυσης προβλημάτων Προγραμματισμού Δράσης με υποσύνολο στόχων-ενεργειών

Στην παρούσα διπλωματική εργασία έγινε ένας συνδυασμός μεταξύ των προσεγγίσεων που υπάρχουν για την επίλυση των προβλημάτων προγραμματισμού δράσης. Οι δύο προσεγγίσεις ήταν η ευρετική αναζήτηση που χρησιμοποιεί το LAMA και η μετατροπή σε πρόβλημα ικανοποιησιμότητας προτασιακής λογικής που χρησιμοποιεί το SATPLAN. Τα δύο εργαλεία συνδυάστηκαν ως εξής: Το SATPLAN δέχεται σαν είσοδο του το παραγόμενο πλάνο του LAMA όμως με ένα υποσύνολο των στόχων του παραγόμενου πλάνου του LAMA. Η τεχνική αυτή είχε ως στόχο την συμπίεση των τελικών δράσεων του αρχείου των ενεργειών. Δηλαδή, να μειωθεί ο συνολικός αριθμός των ενεργειών που υπάρχουν σε ένα αρχείο ενεργειών.

Σε αυτή την ενότητα γίνεται ένας συνδυασμός μεταξύ των δύο προσεγγίσεων επίλυσης προβλημάτων Προγραμματισμού Δράσης. Πριν να εκτελεστεί το SATPLAN, ο χρήστης προσδιορίζει το υποσύνολο των γραμμών των ενεργειών στο οποίο θέλει να συμπιέσει το αρχείο των ενεργειών, όπου το αρχείο ενεργειών είναι η έξοδος του συστήματος LAMA. Αφού προσδιοριστεί το υποσύνολο των γραμμών των ενεργειών τότε εκτελείται το πρόγραμμα για την προσαρμογή του αρχείου προβλήματος όπου σε αυτό διαγράφονται ή προστίθενται καταστάσεις. Το πρόγραμμα αυτό δέχεται σαν είσοδο ένα αρχείο πεδίου ορισμού (domain), το αρχείο προβλήματος (problem) και την έξοδο του LAMA για τα αρχεία αυτά, όπου σε αυτή την περίπτωση είναι το υποσύνολο των στόχων-ενεργειών. Μετά από τα παραπάνω εκτελείται το σύστημα SATPLAN. Το σύστημα SATPLAN δέχεται σαν είσοδο ένα αρχείο πεδίου ορισμού (domain) και το νέο αρχείο προβλήματος (problem) το οποίο έχει τροποποιηθεί με την αφαίρεση ή πρόσθεση καταστάσεων. Στη συνέχεια, ο

SATPLAN εκτελείται με βάση τα αρχεία που δίνονταν στην είσοδο. Όταν τελειώσει η εκτέλεση του SATPLAN η έξοδος του SATPLAN αντικατάστησε το κομμάτι του υποσυνόλου. Επίσης, γινόταν καταγραφή του χρόνου ολοκλήρωσης του συστήματος, ο αριθμός των προηγούμενων ενεργειών και ο νέος αριθμός των ενεργειών του αρχείου με αυτό τον τρόπο. Οι μετρήσεις που έγιναν για αυτό το κομμάτι είναι με βάση των αριθμών των γραμμών. Επίσης, στο εργαλείο SATPLAN υπάρχουν ενέργειες οι οποίες μπορούν να εκτελούνται παράλληλα όμως σε αυτό το κομμάτι μετριόνταν σαν γραμμή η κάθε ενέργεια.

Άλλες μετρικές οι οποίες πάρθηκαν στην ενότητα αυτή για σύγκριση ήταν το αποτέλεσμα του εργαλείου PSP-H το οποίο δεχόταν σαν είσοδο ο SATPLAN μετά από την επιλογή του υποσυνόλου του αρχείου ενεργειών και από την τροποποίηση του αρχείου προβλήματος.

Οι μετρήσεις για αυτή την πειραματική αξιολόγηση έγιναν με βάση τα προβλήματα Elevator, Transport, Visit, Storage και Barman. Επίσης, στις μετρήσεις δόθηκε χρονικό περιθώριο (timeout) για την ολοκλήρωση των συστημάτων μία ώρα. Δηλαδή, αν το σύστημα δεν επιλύσει το πρόβλημα μέσα στο περιθώριο της μιας ώρας τότε θεωρείται ότι το πρόβλημα δεν επιλύεται.

Αυτή η περίπτωση συμβολίζεται μέσα στους πίνακες με τον χαρακτήρα (-). Τα αποτελέσματα περιγράφονται στους πίνακες 7.3.1-7.3.5.

Μέτρηση χρόνου και επιπέδων για το συνδυασμό των δύο συστημάτων (SATPLAN & PSP-H) Domain: Elevator (Υποσύνολο 25%-75% μετά 0%-50% και τέλος 50%-100%)			
Domain-Problem	Προηγούμενα Επίπεδα (levels)	Νέα Επίπεδα (levels)	Χρόνος Επίλυσης (sec)
p01	8	11	192
p02	11	12	195
p03	6	7	226
p04	13	14	200
p05	8	11	599
p06	16	17	594
p07	16	21	240
p08	18	20	464
p09	15	17	637
p10	25	29	1855

Πίνακας 7.3.1 Πρόβλημα Elevator για όλη την διαδικασία

Μέτρηση χρόνου και επιπέδων για το συνδυασμό των δύο συστημάτων (SATPLAN & PSP-H) Domain: Transport (Υποσύνολο 25%-75% μετά 0%-50% και τέλος 50%-100%)			
Domain-Problem	Προηγούμενα Επίπεδα (levels)	Νέα Επίπεδα (levels)	Χρόνος Επίλυσης (sec)
p01	6	5	201
p02	21	11	305
p03	36	14	150
p04	57	17	320
p05	72	-	-
p06	88	-	-
p07	107	-	-
P11	11	10	199
P12	30	18	250
p13	41	27	294

Πίνακας 7.3.2 Πρόβλημα Transport για όλη την διαδικασία

Μέτρηση χρόνου και επιπέδων για το συνδυασμό των δύο συστημάτων (SATPLAN & PSP-H) Domain: Visit (Υποσύνολο 25%-75% μετά 0%-50% και τέλος 50%-100%)			
Domain-Problem	Προηγούμενα Επίπεδα (levels)	Νέα Επίπεδα (levels)	Χρόνος Επίλυσης (sec)
p04-full	19	15	342
p04-half	13	13	357
p05-full	33	23	249
p05-half	22	20	483
p06-full	47	35	560
p06-half	30	24	456
p07-full	64	-	-
p07-half	61	38	745
p08-full	80	-	-
p08-half	66	48	487
p09-full	110	-	-
p09-half	97	-	-

Πίνακας 7.3.3 Πρόβλημα Visit για όλη την διαδικασία

Μέτρηση χρόνου και επιπέδων για το συνδυασμό των δύο συστημάτων (SATPLAN & PSP-H) Domain: Storage (Υποσύνολο 25%-75% μετά 0%-50% και τέλος 50%-100%)			
Domain-Problem	Προηγούμενα Επίπεδα (levels)	Νέα Επίπεδα (levels)	Χρόνος Επίλυσης (sec)
p01	3	4	537
p02	3	5	680
p03	3	5	513
p04	10	8	520
p05	5	6	378
p06	5	6	380
p07	16	15	134
P08	9	9	378
P09	6	8	229
p10	20	19	1844

Πίνακας 7.3.4 Πρόβλημα Storage για όλη την διαδικασία

Μέτρηση χρόνου και επιπέδων για το συνδυασμό των δύο συστημάτων (SATPLAN & PSP-H) Domain: Barman (Υποσύνολο 25%-75% μετά 0%-50% και τέλος 50%-100%)			
Domain-Problem	Προηγούμενα Επίπεδα (levels)	Νέα Επίπεδα (levels)	Χρόνος Επίλυσης (sec)
p02	33	56	754
p03	35	58	102
p04	47	76	1800
p05	45	76	1820
p06	48	76	754
p07	46	74	1844
P08	69	83	2083

Πίνακας 7.3.5 Πρόβλημα Barman για όλη την διαδικασία

Μέτρηση χρόνου και επιπέδων για το συνδυασμό των δύο συστημάτων (SATPLAN & LAMA) Domain: Elevator (Υποσύνολο 25%-75% μετά 0%-50% και τέλος 50%-100%)			
Domain-Problem	Προηγούμενα Επίπεδα (levels)	Νέα Επίπεδα (levels)	Χρόνος Επίλυσης (sec)
p01	22	43	180
p02	24	65	184
p03	24	46	720
p04	36	78	191
p05	32	62	241
p06	48	84	890

Πίνακας 7.3.6 Πρόβλημα Elevator για όλη την διαδικασία

Μέτρηση χρόνου και επιπέδων για το συνδυασμό των δύο συστημάτων (SATPLAN & LAMA) Domain: Transport (Υποσύνολο 25%-75% μετά 0%-50% και τέλος 50%-100%)			
Domain-Problem	Προηγούμενα Επίπεδα (levels)	Νέα Επίπεδα (levels)	Χρόνος Επίλυσης (sec)
p01	6	15	310
p02	21	38	461
p03	36	58	432
p04	57	80	456
P11	11	35	199
P12	30	50	134

Πίνακας 7.3.7 Πρόβλημα Transport για όλη την διαδικασία

Μέτρηση χρόνου και επιπέδων για το συνδυασμό των δύο συστημάτων (SATPLAN & LAMA) Domain: Visit (Υποσύνολο 25%-75% μετά 0%-50% και τέλος 50%-100%)			
Domain-Problem	Προηγούμενα Επίπεδα (levels)	Νέα Επίπεδα (levels)	Χρόνος Επίλυσης (sec)
p04-full	19	28	608
p04-half	13	22	343
p05-full	33	43	924
p05-half	22	35	166
p06-full	47	-	-
p06-half	30	42	424
p07-half	61	67	600
p08-half	66	79	1200

Πίνακας 7.3.8 Πρόβλημα Visit για όλη την διαδικασία

Μέτρηση χρόνου και επιπέδων για το συνδυασμό των δύο συστημάτων (SATPLAN & LAMA) Domain: Storage (Υποσύνολο 25%-75% μετά 0%-50% και τέλος 50%-100%)			
Domain-Problem	Προηγούμενα Επίπεδα (levels)	Νέα Επίπεδα (levels)	Χρόνος Επίλυσης (sec)
p01	3	5	73
p02	3	5	107
p03	3	5	101
p04	8	15	274
p05	9	19	356
p06	9	18	231

Πίνακας 7.3.9 Πρόβλημα Storage για όλη την διαδικασία

Μέτρηση χρόνου και επιπέδων για το συνδυασμό των δύο συστημάτων (SATPLAN & LAMA) Domain: Barman (Υποσύνολο 25%-75% μετά 0%-50% και τέλος 50%-100%)			
Domain-Problem	Προηγούμενα Επίπεδα (levels)	Νέα Επίπεδα (levels)	Χρόνος Επίλυσης (sec)
p02	46	84	434
p03	51	82	1833
p04	69	102	1844
p05	66	103	1844
p06	60	108	2083

Πίνακας 7.3.10 Πρόβλημα Barman για όλη την διαδικασία

Κεφάλαιο 8

Συμπεράσματα

8.1 Συμπεράσματα υλοποίησης

Κατά τη διάρκεια της πρώτης φάσης της παρούσας διπλωματικής εργασίας έγινε εξοικείωση με τα εργαλεία LAMA2011 και SATPLAN2006 με τον κωδικοποιητή SMP (SAT-MAX-PLAN). Για τον λόγο αυτό έχουν παρθεί μερικές μετρικές σχετικά με τα αποτελέσματα των δύο αυτών εργαλείων για να εξαχθούν κάποια συγκεκριμένα συμπεράσματα καθώς επίσης και συγκρίσεις μεταξύ των εργαλείων αυτών.

Συμπεράσματα βάση τους πίνακες 7.1.1,7.1.2,7.2.1,7.2.2:

Το αποτέλεσμα των δύο αυτών εργαλείων είναι ένα αρχείο ενεργειών ως έξοδος. Μελετήθηκε ο αριθμός των ενεργειών των συγκεκριμένων αρχείων που έχουν σαν έξοδο τα εργαλεία αυτά. Όπως παρατηρείται από τους πιο πάνω πίνακες ο αριθμός των ενεργειών που επιστρέφει το εργαλείο SATPLAN είναι λιγότερες από ότι οι ενέργειες που επιστρέφει το εργαλείο LAMA. Αυτό οφείλεται στο γεγονός ότι ο SATPLAN έχει ως στόχο να βρίσκει λύσεις με το μικρότερο μήκος. Το σύστημα αυτό κατασκευάζει αρχικά ένα γράφο μέχρι ένα επίπεδο και μετά μεταφράζει τους περιορισμούς οι οποίοι προκύπτουν σε Συζευκτική Κανονική Μορφή (CNF) για να μπορεί να δώσει τους περιορισμούς αυτούς σε ένα προτασιακό επιλυτή. Αντίθετα, ο LAMA υπολογίζει στην αρχή μια γρήγορη λύση με τη χρήση ενός αποδοτικού αλγορίθμου και μετά γίνεται αναζήτηση για μικρότερου μήκους πλάνα καλώντας διαδοχικές εκτελέσεις του A* αλγορίθμου με μειωμένα τα βάρη.

Παρατηρήθηκε επίσης ότι το εργαλείο SATPLAN μπορεί να εκτελέσει περισσότερες από μία ενέργειες παράλληλα ενώ το εργαλείο LAMA δεν έχει τη δυνατότητα αυτή. Ακόμη, το εργαλείο LAMA χρειάζεται λιγότερο χρόνο για τη εξαγωγή του αρχείου εξόδου σε σχέση με το εργαλείο SATPLAN. Αυτό οφείλεται στο γεγονός ότι το εργαλείο SATPLAN πρέπει να δημιουργήσει ένα γράφο για την εξαγωγή των αποτελεσμάτων του και χρειάζεται περισσότερο χρόνο ο γράφος για να κάνει την επεξεργασία των καταστάσεων η οποία απαιτείται. Εν αντιθέσει, το εργαλείο LAMA δεν δημιουργεί γράφο και χρειάζεται επομένως λιγότερος χρόνος για την εξαγωγή των αποτελεσμάτων του.

Συμπεράσματα βάση τους πίνακες 7.3.1, 7.3.2, 7.3.3, 7.3.4, 7.3.5, 7.3.6, 7.3.7, 7.3.8, 7.3.9, 7.3.10:

Τα αποτελέσματα με βάση τους πίνακες αυτούς αποδεικνύει ότι το πρόγραμμα το οποίο έχει υλοποιηθεί έχει επιτύχει τους στόχους του. Με τις διάφορες τεχνικές οι οποίες έχουν υλοποιηθεί και έχουν περιγραφεί πιο πάνω έχει επιτευχθεί να γίνει μείωση του συνολικού αριθμού των ενεργειών που υπάρχουν σε ένα αρχείο ενεργειών που δίνει σαν αποτέλεσμα το εργαλείο LAMA. Επιπρόσθετα, όσο αφορά τον χρόνο στο οποίο ένα πρόβλημα Προγραμματισμού Δράσης χρειάζεται να επιλυθεί εξαρτάται σε μεγάλο βαθμό ο χρόνος επίλυσης του από το εργαλείο SATPLAN.Η μέθοδος η οποία χρησιμοποιήθηκε στην οποία ένα αρχείο ενεργειών διασπάται σε μικρότερα μέρη για την επίλυση του, είναι πιο αποδοτική τόσο στο συνολικό χρόνο όσο και στο συνολικό αριθμό των ενεργειών το οποίο χρειάζονται για να επιλυθούν τα προβλήματα αυτά.

Επίσης, χρησιμοποιώντας την μέθοδο η οποία μπορεί να παρομοιαστεί με ένα ενδιάμεσο σταθμό, δηλαδή για το κομμάτι του υποσυνόλου των γραμμών του αρχείου ενεργειών 0%-50% στην οποία το εργαλείο SATPLAN εκτελείται δύο φορές, έχει ως αποτέλεσμα καλύτερο χρόνο από ότι να εκτελεστεί το εργαλείο SATPLAN μία φορά για ολόκληρο το Πρόβλημα Προγραμματισμού Δράσης. Ακόμη, με την μέθοδο αυτή ο συνολικός αριθμός των ενεργειών στο αρχείο ενεργειών είναι μικρότερος από ότι του εργαλείου LAMA.

8.2 Μελλοντικές Επεκτάσεις

Θα μπορούσαν να υπάρχουν πολλές μελλοντικές επεκτάσεις καθώς και βελτιστοποιήσεις στην παρούσα διπλωματική εργασία. Οι βελτιστοποιήσεις οι οποίες θα μπορούσαν να γίνουν είναι η χρήση άλλων δομών για να επιτευχθούν καλύτερα αποτελέσματα τόσο σε χρόνο όσο

και σε μη σπατάλη της αχρησιμοποίητης μνήμης. Ακόμη μία βελτιστοποίηση η οποία θα μπορούσε να γίνει είναι να μπορεί να δέχεται το πρόγραμμα όλες τις μορφές των αρχείων που έχουν τα Προβλήματα Προγραμματισμού Δράσης χωρίς να χρειάζεται η αρχική επεξεργασία του αρχείου για να μπορεί να λειτουργεί σωστά.

Επιπρόσθετα, η υλοποίηση του προγράμματος αυτού θα μπορούσε να γίνει σε άλλη γλώσσα προγραμματισμού και ποιο συγκεκριμένα σε μια αντικειμενοστραφή γλώσσα προγραμματισμού, όπως για παράδειγμα η JAVA. Αυτές οι γλώσσες παρέχουν περισσότερες δυνατότητες και διευκολύνσεις στον προγραμματιστή γιατί έχουν πολλές βιβλιοθήκες και πολλές συναρτήσεις υλοποιημένες. Με αυτό τον τρόπο ο προγραμματιστής δεν χρειάζεται να γράφει συναρτήσεις μιας και οι συναρτήσεις αυτές ήδη είναι υλοποιημένες.

Βιβλιογραφία

[1] H. Kautz , Bart Selman, “SatPlan: Planning as Satisfiability”

[2] A.Sideris, Y.Dimopoulos, “Constraint Propagation in Propositional Planning”, Proceedings of the Twentieth International Conference on Automated Planning and Scheduling (ICAPS 2010)

[3] Σημειώσεις μαθήματος Τεχνητής Νοημοσύνης ΕΠΛ341, Διάλεξη $17^{η}$ και $18^{η}$

[4] Σημειώσεις μαθήματος Τεχνητής Νοημοσύνης ΕΠΛ341, Διάλεξη $1^{η}$

[5] “Introduction to AI”, Prof. Richard Zanibbi, RIT

[6] “Review of Classical Planning & HTN Planning”, Stanford University

[7] Σημειώσεις μαθήματος
«http://aiserver.uom.gr/Courses/ArtificialIntelligence/Slides/Chapter9.pdf»

Παράρτημα Α

Κύρια συνάρτηση:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "h1.h"
#include "h2.h"
#include <sys/time.h>

#include <unistd.h>
int main(int argc, char *argv[]){

  FILE *domain,*problem,*sas;
  FILE *out=fopen("output1.txt","w");
  FILE *newi;
  FILE *in;
  FILE *in1;
FILE *new;
FILE *out2;
FILE *newproblem=fopen("provlima.txt","w");
FILE *newproblem2=fopen("provlima2.txt","w");
FILE *newproblem3=fopen("provlima3.txt","w");
FILE *newproblem4=fopen("provlima4.txt","w");
FILE *newproblem5=fopen("provlima5.txt","w");
int energies[500];
int i=0;
int min;
int max;
int outg=0;
  int grammes=0;
int flag=0;
char str[500];
int ep;
int ep1;
int num=0;
char *s1="mera.pddl";
char *s2="merc.pddl";
char *s3="merb.pddl";

struct timeval end,start;

gettimeofday(&start,NULL);


 if(argc<4){
   printf("Error in arguments!\n");
   exit(-1);
 }
```

```
 domain=fopen(argv[1],"r");


 if(domain==NULL)
   {
     printf("Error..mi dinato anoigma tou arxeiou\n");
     exit(-1);
   }
   problem=fopen(argv[2],"r");
  if(problem==NULL)
   {
     printf("Error..mi dinato anoigma tou arxeiou\n");
     exit(-1);
   }



 sas=fopen(argv[3],"r");

 if(sas==NULL)
   {
     printf("Error..mi dinato anoigma tou arxeiou\n");
     exit(-1);
   }

 rewind(sas);

rename(argv[1],"domain.pddl");

grammes=metrimagrammon(sas);
antigrafi_provlimatos(problem,newproblem);
antigrafi_provlimatos(problem,newproblem2);
antigrafi_provlimatos(problem,newproblem3);
antigrafi_provlimatos(problem,newproblem4);
antigrafi_provlimatos(problem,newproblem5);
fclose(problem);
fclose(newproblem);
fclose(newproblem2);
fclose(newproblem3);
fclose(newproblem4);
///periptosi gia 25%-75%
xorismos_arxeion(out,sas,grammes,&min,&max,out2);
out=fopen("output1.txt","r");
outg=metrimagrammon(out);
//printf("outg ine %d\n",outg);
if(outg!=0 ){
out=fopen("output1.txt","r");

merosa(domain,problem,out,argv[2],argv[3],newi,0,0);
out2=fopen("output2.txt","r");
merosa(domain,newproblem,out2,"provlima.txt",argv[3],newi,0,1);
```

```
fclose(problem);
meros("provlima.txt",newproblem);
meros("newinit.txt",newi);
teliko_arxeio_probl(newi,problem,argv[2],s1);}
else{
out2=fopen("output2.txt","r");
merosa(domain,newproblem,out2,"provlima.txt",argv[3],newi,1,0);
meros2(s1,newproblem,"provlima.txt");

}
system("./satplan -encoding 5 -domain domain.pddl -problem mera.pddl");

sas=fopen(argv[3],"r");
num++;
prosarmogi_arxeiou(in,sas,new,min,max,argv[3],num);
///periptosi gia 0%-50%
remove("output2.txt");
sas=fopen(argv[3],"r");
grammes=metrimagrammon(sas);
xorismos_arxeion50(out,sas,grammes,&min,&max);
out=fopen("output1.txt","r");
merosa(domain,newproblem2,out,"provlima2.txt","provlima2.txt",newi,0,1);
//fclose(newproblem2);
//fclose(newi);
meros2(s2,newproblem2,"provlima2.txt");
meros2(s3,newi,"newinit.txt");
remove("newinit.txt");
//teliko_arxeio_probl2(newproblem2,"provlima2.txt",s2);


system("./satplan -encoding 5 -domain domain.pddl -problem merb.pddl");
system("./satplan -encoding 5 -domain domain.pddl -problem merc.pddl");
sas=fopen(argv[3],"r");
num++;
prosarmogi_arxeiou2(in,in1,sas,new,argv[3],&flag);
//}
sas=fopen(argv[3],"r");
teliko_arxeio(sas,argv[3],flag);

remove("output2.txt");
remove("output1.txt");
remove("newinit.txt");
remove("provlima.txt");
remove("provlima2.txt");
remove("provlima3.txt");
remove("provlima4.txt");
rename("provlima5.txt",argv[2]);
fclose(newproblem5);
rename("domain.pddl",argv[1]);
gettimeofday(&end,NULL);
printf("O xronos einai : %.2lu sec\n",(end.tv_usec-start.tv_usec)/1000);
  return 0;}
```