Ατομική Διπλωματική Εργασία

ΕΥΡΕΤΙΚΕΣ ΜΕΘΟΔΟΙ ΣΥΝΟΛΟΥ ΑΠΑΝΤΗΣΕΩΝ ΣΤΗΝ ΕΠΙΛΥΣΗ ΠΡΟΒΛΗΜΑΤΩΝ ΔΡΑΣΗΣ

Στέλιος Στεφανή

Πανεπιστήμιο Κύπρου

Τμήμα Πληροφορικής

Μάιος 2020

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Ευρετικές Μέθοδοι Συνόλου Απαντήσεων στην Επίλυση Προβλημάτων Προγραμματισμού Δράσης

Στέλιος Στεφανή

Επιβλέπων Καθηγητής
Γιάννης Δημόπουλος

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

Μάιος 2020

Ευχαριστίες

Θα ήθελα να ευχαριστήσω θερμά τον Καθηγητή του Τμήματος Πληροφορικής κ Γιάννη Δημόπουλο για την εμπιστοσύνη που μου έδειξε στην εκπόνηση της παρούσας διπλωματικής εργασίας καθώς και για όλες τις υποδείξεις και συμβουλές του καθ' όλα τα στάδια της πειραματικής έρευνας.

Περίληψη

Αυτή η εργασία αναφέρεται στη χρήση ευρετικών μεθόδων για την επίλυση Προβλημάτων Δράσης μέσω Προγραμματισμού Συνόλου Απαντήσεων. Υπάρχει μια μικρή εισαγωγή στους τομείς Προγραμματισμός Συνόλου Απαντήσεων , Προβλήματα Δράσης και Ευρετικές Μέθοδοι και έπειτα αναφέρεται πως έγινε συνδυασμός των πιο πάνω εννοιών για τη διεκπεραίωση πειραματικής μελέτης η οποία είχε σκοπό την αύξηση αποδοτικότητας επίλυσης Προβλημάτων Δράσης μέσω Προγραμματισμού Συνόλου Απαντήσεων με τη χρήση Ευρετικών Μεθόδων. Μέσω της διαδικασίας αυτής υπήρξε αύξηση αποδοτικότητας στον αριθμό των επιλυμένων προβλημάτων μέχρι και 16% και αύξηση του αριθμού των προβλημάτων που λύθηκαν κάτω από όριο 900 δευτερολέπτων κατά 10.5%

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

1.1 Εισαγωγή

Εδώ και πολλά χρόνια οι επιστήμονες της πληροφορικής ασχολούνται με τις προκλήσεις που υπάρχουν στη βελτιστοποίηση τεχνικών και μεθόδων που αφορούν το λογικό προγραμματισμό. Μέσω διαφόρων πειραματικών μελετών προσπαθούν να ανακαλύψουν νέους τρόπους για την επιτάχυνση και αύξηση της ακριβείας όλων των υπαρχόντων μεθόδων. Δημιουργούν συνεχώς αλγορίθμους και στρατηγικές για την επιτάχυνση της επίλυσης προβλημάτων αλλά υπάρχουν ακόμη αρκετά περιθώρια βελτίωσης. Ένας από τους τομείς που γίνεται έρευνα για βελτιστοποίηση των λύσεων είναι η εύρεση πλάνων. Με τη χρήση Προγραμματισμού Συνόλου Απαντήσεων (ΠΣΑ) μπορούν να λυθούν τέτοια προβλήματα αλλά υπάρχει πρόβλημα στο θέμα της αποδοτικότητας . Μια μέθοδος που τείνει να δίνει καλά αποτελέσματα, τόσο σε θέμα χρόνου όσο και σε ακρίβεια επίλυσης ,στο τομέα της εύρεσης πλάνων, είναι η χρήση ευρετικών. Αρκετοί επιλυτές κάνουν χρήση ευρετικών μεθόδων έτσι ώστε να βελτιώσουν το χρόνο εκτέλεσης τους. Τα ευρετικά μπορούν να μειώσουν δραματικά το χρόνο εξερεύνησης αφού προσθέτουν στο πλάνο επίλυσης νέους περιορισμούς για την επιλογή βημάτων. Μπορεί να γίνει χρήση ευρετικών μεθόδων χωρίς να αλλαχθεί η υλοποίηση του επιλυτή ή των υπάρχων ευρετικών του. Αυτό μπορεί να επιτευχθεί με το συνδυασμό χρήσης ΠΣΑ και γλώσσας σεναρίων (script languages) ή προστακτικού προγραμματισμού.

1.2 Προεπισκόπηση Αναφοράς

Η εργασία αυτή αναφέρεται στη πειραματική μελέτη διαφόρων ευρετικών μεθόδων για την επίλυση Προβλημάτων Δράσης μέσω Προγραμματισμού Συνόλου Απαντήσεων. Χρησιμοποιήθηκαν στη πειραματική μελέτη τα ακόλουθα εργαλεία, ο επιλυτής ΠΣΑ clingo, ο μεταφραστής plasp ο οποίος παράγει τα δεδομένα εισόδου για τον επιλυτή σε μορφή ΠΣΑ από PDDL μορφή και η γλώσσα προγραμματισμού Python για την δημιουργία αρχείων αποτελεσμάτων, επεξεργασία δεδομένων και για τις κλήσεις του επιλυτή. Στη συνεχεία υπάρχουν γενικά στοιχεία για τις διάφορες μεθόδους και εργαλεία που χρησιμοποιήθηκαν καθώς και τα αποτελέσματα της έρευνας. Συγκεκριμένα, στο 2ο κεφάλαιο υπάρχουν γενικά στοιχεία του ΠΣΑ και του επιλυτή clingo, στο 3ο κεφάλαιο καταγράφεται πως μπορούν να επιλυθούν προβλήματα δράσης μέσω του ΠΣΑ, στο 4ο κεφάλαιο υπάρχουν στοιχεία για τις ευρετικές μεθόδους και για το πως οι ευρετικές μέθοδοί μπορούν να βοηθήσουν στην επίλυση προβλημάτων δράσης μέσω ΠΣΑ, στο 5ο κεφάλαιο γίνεται αναφορά στη μεθοδολογία που ακολουθήθηκε καθώς και στο πως εκτελείται η πειραματική μελέτη, στο 6ο κεφάλαιο υπάρχουν τα αποτελέσματα της πειραματικής μελέτης και στο 7ο κεφάλαιο υπάρχουν τα συμπεράσματα που εξήχθησαν από αυτή την έρευνα και μελλοντικές πειραματικές μελέτες που μπορούν να ακολουθήσουν .

Κεφάλαιο 2

Προγραμματισμός Συνόλου Απαντήσεων και Προγραμματισμός Δράσης

2.1 Γενικές πληροφορίες για τον Προγραμματισμό Συνόλου Απαντήσεων

Ο Προγραμματισμός Συνόλου Απαντήσεων - ΠΣΑ (Answer Set Programming - ASP) είναι μια μορφή δηλωτικού προγραμματισμού (declarative programming) και λογικού προγραμματισμού η οποία ασχολείται με δύσκολα προβλήματα εύρεσης και κυρίως με NP-hard προβλήματα. Είναι βασισμένος στα stable model semantics του λογικού προγραμματισμού τα οποία δημιουργήθηκαν με βάση τις ιδέες των autoepistemic logic και default logic. Χρησιμοποιεί κανόνες που μοιάζουν με κανόνες Prolog αλλά μέσω διαφορετικών υπολογιστικών μεθόδων. Ο ΠΣΑ είναι αρκετά χρήσιμος σε knowledge-intensive εφαρμογές. Οι αλγόριθμοι που χρησιμοποιούνται για τον ΠΣΑ είναι μετατροπές του αλγορίθμου PDLL και έχουν ομοιότητες με αποδοτικούς SAT επιλυτές. Η διαφορά του ΠΣΑ από άλλες μορφές λογικού προγραμματισμού βρίσκεται στην αναπαράσταση των λύσεων. Ο ΠΣΑ χρησιμοποιεί σύνολα απαντήσεων (answer sets) για να αναπαριστά τις λύσεις του ενώ οι υπόλοιπες μορφές λογικού προγραμματισμού χρησιμοποιούν σύνολα αντικαταστάσεων (answer substitutions). Είναι δυνατή επίσης η χρήση περιορισμών δίνοντας έτσι αρκετή ευελιξία στα ΠΣΑ προγράμματα. Η κύρια ιδέα του ΠΣΑ είναι η δημιουργία ενός προγράμματος στο οποίο περιγράφεται ένα πρόβλημα χωρίς να χρησιμοποιηθούν μεταβλητές, δηλαδή με τη χρήση σταθερών μόνο.

2.2 Παραδείγματα ΠΣΑ

Ο ΠΣΑ έχει υιοθετήσει εν μέρη τον τρόπο δήλωσης των κανόνων του από τη γλώσσα Proloq. Για παράδειγμα ο κανόνας **p :- q.** θα δημιουργήσει το Σύνολο Απαντήσεων {p , q} εάν το q είναι αληθές στο πρόγραμμα ,ο κανόνας **q :- not r.** θα δημιουργήσει το Σύνολο Απαντήσεων {q} αν το r δεν είναι αληθές στο πρόγραμμα και ο κανόνας **{s,t} :- p.** θα δημιουργήσει τα Σύνολο Απαντήσεων {s}, {t},{s , t} εάν το p είναι αληθές στον κόσμο. Αν συνδυάσουμε και τους 3 κανόνες μπορούν να παραχθούν διάφορα Σύνολα Απαντήσεων όπως {q} αν όλα τα υπόλοιπα άτομα είναι ψευδές στο κόσμο, {r} αν μόνο το r είναι αληθές στον κόσμο και {p ,q ,s ,t} ένα το p είναι αληθές. Μπορούν να χρησιμοποιηθούν επίσης κανόνες που περιορίζουν τον αριθμό των ατόμων που θα επιλεχθούν από κάποιο σύνολο. Ο κανόνας **1 {s,t} :- p.** περιορίζει το αποτέλεσμα να έχει τουλάχιστον ένα από τα άτομα του συνόλου εάν ισχύει το p στον κόσμο. Θα παραχθούν δηλαδή τα Σύνολα Απαντήσεων {s}, {t} και {s ,t} εφόσον ισχύει το p. Αναλόγως ο κανόνας **{s,t} 1 :- p.** επιτρέπει το πολύ ένα άτομο να επιλεχθεί από το σύνολο εάν ισχύει το p στο κόσμο. Σε αυτή τη περίπτωση τα Σύνολα Απαντήσεων είναι {}, {s} και {t}. Μπορεί δηλαδή να μην επιλεχθεί κανένα από τα άτομα του συνόλου και να πάρουμε το κενό σύνολο ως Σύνολο Απαντήσεων . Υπάρχουν επίσης κανόνες που δρουν ως περιορισμοί. Ο κανόνας **:- s, not t.** αποτρέπει το s να υπάρχει στο Σύνολο Απαντήσεων εάν δεν υπάρχει το t. Υπάρχουν δηλαδή δύο πιθανά Σύνολο Απαντήσεων για αυτό το παράδειγμα, {} και {s, t}.

Οι μεταβλητές σε ένα ΠΣΑ πρόγραμμα απλοποιούνται μέσω της διαδικασίας σταθεροποίησης (grounding). Οι ακόλουθοί κανόνες όταν περάσουν από τη διαδικασία σταθεροποίησης θα δημιουργήσουν το αντίστοιχο πρόγραμμα χωρίς μεταβλητές.

Αρχικό πρόγραμμα

p(a). p(b). p(c). --- άτομα που υπάρχουν στον κόσμο

q(X): - p(X).

2 {r(X): p(X)}.

Πρόγραμμα μετά από σταθεροποίηση

p(a). p(b). p(c). --- άτομα που υπάρχουν στον κόσμο

q(a): - p(a).

q(b): - p(b).

q(c): - p(c).

2 {r(a), r(b), r(c)}.

Σε κάποιες περιπτώσεις χρειάζεται να δηλωθούν αρκετά άτομα του ίδιου είδους. Αυτό μπορεί να επιτευχθεί με τη χρήση συνόλων. Για παράδειγμα ο κανόνας **vertex(1..99).** θα δημιουργήσει τα άτομα **vertex(1). vertex(2) … vertex(98). vertex(99).** Εάν ο αριθμός των κανόνων που πρέπει να απλοποιηθούν στη διαδικασία σταθεροποίησης είναι αρκετά μεγάλος τότε μπορεί να υπάρξει κάποια καθυστέρηση μέχρι να αφαιρεθούν όλες οι μεταβλητές και να δημιουργηθεί το εκτελέσιμο ΠΣΑ πρόγραμμα.

Μέσω του ΠΣΑ μπορούν να λυθούν πολύπλοκα προβλήματα σε λίγο χρόνο. Το πρόβλημα των ν βασιλισσών μπορεί να λυθεί με την πιο κάτω υλοποίηση.

1{queen(I,1..n)}1:-I=1..n.
1{queen(1..n,J)}1:-J=1..n.
:-2{queen(D-J,J)},D=2..2*n.
:-2{queen(D+J,J)},D=1-n..n-1.

Οι πρώτοι δύο περιορισμοί δείχνουν πως μόνο μια βασίλισσα μπορεί να βρίσκεται σε κάθε γραμμή και στήλη ενώ οι άλλοι δύο περιορισμοί καλύπτουν τον περιορισμό των διαγώνιων. Έτσι μέσω μιας απλής υλοποίησης μπορεί να λυθεί ένα np complete πρόβλημα σε πολύ λίγο χρόνο. Αν βέβαια ο αριθμός του ν είναι αρκετά μεγάλος τότε ο επιλυτής μπορεί να κάνει αρκετή ώρα να βρει τη λύση

2.3 Επιλυτής Clingo

Για την επίλυση ΠΣΑ μπορεί να γίνει χρήση του επιλυτή clingo ο οποίος εκτελεί τη διαδικασία σταθεροποίησης, δηλαδή μετατρέπει το αρχικό πρόγραμμα σε ένα πρόγραμμα χωρίς μεταβλητές αλλά διατηρώντας σταθερά τα Σύνολα Απαντήσεων. Αυτή η διαδικασία μπορεί να είναι αρκετά χρονοβόρα εάν πρέπει να δημιουργηθεί μεγάλος αριθμός κανόνων για αυτό πρέπει να γίνεται χρήση της μόνο όταν είμαστε σίγουροι ότι υπάρχει πεπερασμένος αριθμός κανόνων. Στον clingo υπάρχουν αρκετές βελτιστοποιήσεις έτσι ώστε να μπορεί να βρεθεί η λύση πιο γρήγορα. Είναι κατάλληλο ακόμη για την επίλυση προβλημάτων που αλλάζει η λογική τους συνεχώς. Επιπλέον καλεί την διαδικασία σταθεροποίησης και επίλυσης μόνο μια φορά έτσι ώστε να εκμεταλλευτεί πλήρως τις δυνατότητες των επιλυτών του. Υπάρχει ακόμη η δυνατότητα δημιουργίας υποπρογραμμάτων (subprograms) των οποίων η ενσωμάτωση και η διαδικασία σταθεροποίησης ελέγχονται από τη διαδικαστικό μέρος του επιλυτή. Η διαδικασία που ακολουθεί ο clingo για την επίλυση των προβλημάτων είναι η εξής , αρχικά παράγετε το πεπερασμένο σύνολο κανόνων από το πρόγραμμα που δίνεται σαν είσοδος μέσω της διαδικασίας σταθεροποίησης και έπειτα ο επιλυτής υπολογίζει τα stable models των κανόνων που δημιούργησε η διαδικασία σταθεροποίησης. Ένα ακόμα πλεονέκτημα του clingo είναι ότι είναι εύκολα προσβάσιμο από scripting languages όπως η Python, η οποία έχει έτοιμα πακέτα και APIs για την διευκόλυνση του χρήστη.

2.4 Προβλήματα Δράσης

Τα προβλήματα δράσης στη Τεχνητή Νοημοσύνη αφορούν τα προβλήματα όπου χρειάζεται να δημιουργηθεί ένα πλάνο επίλυσης, μια σειρά κινήσεων - δράσεων που πρέπει να εκτελεστούν, έτσι ώστε να φτάσουμε σε μια κατάσταση που επιθυμούμε. Η σειρά δράσεων που εκτελούνται έτσι ώστε να φτάσουμε από μια αρχική κατάσταση σε μια επιθυμητή τελική κατάσταση κάποιου προβλήματος ονομάζεται πλάνο επίλυσης του προβλήματος. Στα προβλήματα δράσης υπάρχει ο κόσμος ο οποίος περιέχει όλα τα αντικείμενα και τις καταστάσεις στις οποίες βρίσκονται. Οι δράσεις που μπορούν να εκτελεστούν σε κάποιο κόσμο έχουν κάποιες προϋποθέσεις (preconditions) οι οποίες πρέπει να πληρούνται έτσι ώστε να εκτελεστούν οι δράσεις και κάποια αποτελέσματα (effects) στον κόσμο τα οποία είναι μεταβολές στις καταστάσεις στις οποίες βρίσκονται τα αντικείμενα του κόσμου. Μια γλώσσα που χρησιμοποιείται για την επίλυση τέτοιων προβλημάτων είναι η PDDL (Planning Domain Definition Language). Η σύνταξη της PDDL είναι αρκετά απλή. Για τη κωδικοποίηση ενός προβλήματος δράσης στη PDDL χρειάζεται να δημιουργηθεί ένα domain file και ένα problem file. Το domain file περιέχει όλα τα είδη αντικείμενων που μπορούν να υπάρχουν στον κόσμο, τις καταστάσεις στις οποίες μπορούν να βρεθούν τα αντικείμενα και τις δράσεις που μπορούν να εκτελεστούν. Για κάθε δράση υπάρχουν οι παράμετροι - αντικείμενα που εμπλέκονται στην εκτέλεση της, οι προϋποθέσεις (preconditions) που πρέπει να πληρούνται για την εκτέλεση της κίνησης, δηλαδή η κατάσταση στην οποία πρέπει να βρίσκονται τα αντικείμενα που εμπλέκονται, και το αποτέλεσμα (effect) που επιφέρει στις καταστάσεις αντικειμένων του κόσμου όταν εκτελεστεί. Το problem file αποτελείτε από τις αντικείμενα που αποτελούν τον κόσμο, την αρχική κατάσταση του προβλήματος, δηλαδή σε ποια κατάσταση βρίσκεται το κάθε αντικείμενο, και το στόχο – τελική κατάσταση στην οποία πρέπει να βρίσκεται ο κόσμος για να επιλυθεί το πρόβλημα. Θα μπορούσαμε να πούμε ότι το domain αποτελεί τη κωδικοποίηση του κόσμου, τι υπάρχει στο κόσμο και ποιες δράσεις μπορούν να εκτελεστούν, ενώ το problem ένα στιγμιότυπο του κόσμου με ακριβείς αντικείμενα και καταστάσεις.

2.5 Παραδείγματα PDDL

2.5.1 Υλοποίηση Domain file

Για καλύτερη κατανόηση της λειτουργίας της PDDL θα αναλύσουμε τη δομή ενός domain και problem file. Στο Σχήμα 1 υπάρχει η υλοποίηση ενός πολύ απλού domain το οποίο αναφέρεται σε ένα πλέγμα στο οποίο κινείται επάνω ένας πράκτορας (ρομποτ). Στη γραμμή 1 του πιο κάτω κώδικα domain file δηλώνεται το όνομα του domain (**domain grid-visit-all**) το οποίο χρησιμοποιείται από τα problem files για να γίνει η εισαγωγή των εννοιών. Στη γραμμή 2 δηλώνονται οι τύποι των αντικειμένων που υπάρχουν στον κόσμο, σε αυτή τη περίπτωση υπάρχει μόνο το place το οποίο χαρακτηρίζει μια τοποθεσία πάνω σε ένα πλέγμα. Η 3η, 4η και 5η γραμμή αναφέρονται στις καταστάσεις στις οποίες μπορούν να βρίσκονται τα αντικείμενα. Ο κανόνας **connected ?x ?y – place** δείχνει πως οι τοποθεσίες x και y οι οποίες είναι τύπου place είναι γειτονικές, ο κανόνας **at-robot ?x – place** δείχνει πως ο πράκτορας που κινείτε στο πλέγμα βρίσκεται στη τοποθεσία x και ο κανόνας **visited ?x – place** δείχνει πως ο πράκτορας έχει επισκεφτεί τη τοποθεσία x. Οι γραμμές 6, 7, 8 και 9 αναφέρονται στη δήλωση της δράσης που μπορεί να εκτελεστεί στον κόσμο. Στην 6η γραμμή δηλώνεται το όνομα της δράσης, στη συγκεκριμένη περίπτωση **move.** Στην 7η γραμμή καταγράφονται οι παράμετροι – αντικείμενα που επηρεάζουν την εκτέλεση της δράσης. Για τη δράση **move** μας ενδιαφέρει η θέση που βρίσκεται ο πράκτορας τη συγκεκριμένη χρονική στιγμή (**curpos)** και η θέση στην οποία θα μετακινηθεί (**nextpos)**. Στην 8η γραμμή καταγράφονται οι προϋποθέσεις (preconditions) που πρέπει να ισχύουν έτσι ώστε να μπορεί να εκτελεστεί η δράση **move**. Για να μπορέσει ο πράκτορας να μετακινηθεί σε μια νέα τοποθεσία (**nextpos**) αυτή πρέπει να βρίσκεται σε γειτονική θέση (**connected)** με την τοποθεσία που βρίσκεται ο πράκτορας (**curpos**). Τέλος στη 9η γραμμή δηλώνονται η αλλαγές που επιβληθούν στο κόσμο με την εκτέλεση της δράσης. Όταν εκτελεστεί η δράση **move** ο πράκτορας θα πάψει να βρίσκεται στη θέση που ήταν πριν την εκτέλεση (**curpos**), θα μεταφερθεί στη νέα τοποθεσία (**nextpos**) και η νέα τοποθεσία θα περάσει σε κατάσταση **visited** δηλαδή ότι ο πράκτορας την έχει επισκεφθεί.

Σχήμα 1

```
1)    (define (domain grid-visit-all)
2)    (:types       place - object)
3)    (:predicates (connected ?x ?y - place)
4)    (at-robot ?x - place)
5)    (visited ?x - place)  )
6)    (:action move
```

7) **:parameters (?curpos ?nextpos - place)**

8) **:precondition (and (at-robot ?curpos) (connected ?curpos ?nextpos))**

9) **:effect (and (at-robot ?nextpos) (not (at-robot ?curpos)) (visited ?nextpos))))**

2.5.2 Υλοποίηση Problem file

Η υλοποίηση του domain δίνει τη λογική με την οποία λειτουργεί ο κόσμος, ποια αντικείμενα και καταστάσεις υπάρχουν και ποιες δράσεις μπορούν να υλοποιηθούν. Αυτό που απομένει για να έχουμε ένα ολοκληρωμένο πρόγραμμα είναι να δημιουργηθεί ένα πρόβλημα – στιγμιότυπο για το domain. Το problem file αποτελεί το στιγμιότυπο αυτό. Στο Σχήμα 2 φαίνεται η υλοποίηση ενός απλού problem file για το domain πλέγματος. Στη γραμμή 1 δηλώνεται το όνομα του προβλήματος **(problem grid-2)**. Στη γραμμή 2 γίνεται αναφορά στο domain στο οποίο κατατάσσεται το παρών πρόβλημα έτσι ώστε να προστεθούν κατά την επίλυση οι κανόνες που υπάρχουν στο domain μαζί με τα δεδομένα του problem file και να μπορεί να επιλυθεί το πρόβλημα. Οι γραμμές 3 μέχρι 8 δείχνουν πως δηλώνονται τα αντικείμενα. Στο συγκεκριμένο σχήμα δηλώνονται 4 τοποθεσίες οι οποίες αποτελούν το πλέγμα και είναι όλες τύπου place. Έπειτα στη γραμμή 9 μέχρι 19 γίνεται η αρχικοποίηση των καταστάσεων των αντικειμένων. Σε αυτό το παράδειγμα όλες οι τοποθεσίες είναι ενωμένες με τες γειτονικές τους τοποθεσίες, δηλαδή όσες τοποθεσίες έχουν διαφορά 1 σε x ή y. Δημιουργούνται λοιπόν κανόνες που δείχνουν πως η τοποθεσία **loc-x0-y0** είναι ενωμένη (**connected**) με τη τοποθεσία **loc-x0-y1** και ο αντίστροφος κανόνας δηλαδή ότι η τοποθεσία **loc-x0-y1** είναι ενωμένη με τη τοποθεσία **loc-x0-y0** . Ο λόγος που απαιτείτε και η αντίστροφη υλοποίηση είναι ότι αν δεν υπάρχει δεν θα μπορεί να γίνει η δράση **move** από τη $2^{η}$ τοποθεσία στη $1^{η}$ αφού η δράση ελέγχει αν η τοποθεσία που βρίσκεται ο πράκτορας (**curpos**) είναι ενωμένη με την τοποθεσία στην οποία θα μετακινηθεί (**nextpos**). Άρα αν υλοποιηθεί μόνο ο κανόνας **connected loc-x0-y0 loc-x0-y1** αλλά όχι ο κανόνας **connected loc-x0-y1 loc-x0-y0** τότε ο πράκτορας θα μπορεί να μετακινηθεί από τη τοποθεσία (0,0) στη τοποθεσίας (0,1) αλλά όχι από τη τοποθεσία (0,1) στη τοποθεσία (0,0). Τέλος στις γραμμές 20 μέχρι 25 ορίζεται ο στόχος (**goal**) του προβλήματος ο οποίος πρέπει να εκπληρωθεί έτσι ώστε να επιλυθεί το πρόβλημα. Οι γραμμές 22 μέχρι 25 δηλώνουν ότι ο πράκτορας πρέπει να επισκεφθεί κάθε τοποθεσία στο πλέγμα για να λυθεί το πρόβλημα.

Σχήμα 2

1) **(define (problem grid-2)**

2) **(:domain grid-visit-all)**

3) **(:objects**

4) **loc-x0-y0**

5) loc-x0-y1

6) loc-x1-y0

7) loc-x1-y1

8) - place)

9) (:init

10) (at-robot loc-x1-y1)

11) (visited loc-x1-y1)

12) (connected loc-x0-y0 loc-x1-y0)

13) (connected loc-x0-y0 loc-x0-y1)

14) (connected loc-x0-y1 loc-x1-y1)

15) (connected loc-x0-y1 loc-x0-y0)

16) (connected loc-x1-y0 loc-x0-y0)

17) (connected loc-x1-y0 loc-x1-y1)

18) (connected loc-x1-y1 loc-x0-y1)

19) (connected loc-x1-y1 loc-x1-y0))

20) (:goal

21) (and

22) (visited loc-x0-y0)

23) (visited loc-x0-y1)

24) (visited loc-x1-y0)

25) (visited loc-x1-y1))))

2.6 Μεταφραστής plasp

Για να είναι δυνατή η λύση Προβλημάτων Δράσης μέσο ΠΣΑ πρέπει να μετατραπεί το πρόγραμμα PDDL σε μορφή ΠΣΑ. Αυτή η διαδικασία μπορεί να γίνει μεταφράζοντας τη λογική των domain και problem file της PDDL στο αντίστοιχο ΠΣΑ πρόγραμμα που θα επιλύνει το πρόβλημα με τον ίδιο τρόπο. Αυτή η διαδικασία είναι όμως χρονοβόρα και θα χρειάζεται να υλοποιείται νέο πρόγραμμα ΠΣΑ για κάθε διαφορετικό domain και problem. Λύση σε αυτό το πρόβλημα έδωσε η Potassco μέσω του μεταφραστή plasp. Ο plasp παίρνει σαν είσοδο ένα domain file και ένα problem file και τα μεταφράζει σε δεδομένα εισόδου ΠΣΑ μορφής. Η διαδικασία αυτή μπορεί να μεταφράσει domain και problem files τα οποία γίνονται είσοδος αργότερα στο ΠΣΑ πρόγραμμα που περιγράφεται στο Κεφάλαιο 2.7 το οποίο μπορεί να επεξεργαστεί τα μεταφρασμένα δεδομένα και να δώσει λύση στο πρόβλημα μέσο της δημιουργίας ενός πλάνου δράσεων. Στο Σχήμα 1 και 2 περιγράψαμε πως είναι η PDDL μορφή των δεδομένων μας. Ας αναλύσουμε λοιπόν τα δεδομένα που θα επέστρεφε ο μεταφραστής plasp με είσοδο τα αρχεία του Σχήματος 1 και 2. Το πρώτο βήμα που κάνει ο μεταφραστής είναι να δημιουργήσει όλους τους τύπους αντικειμένων που υπάρχουν στο domain, στο Σχήμα 1 υπάρχει μόνο ο τύπος **place** και έτσι δημιουργεί τον αντίστοιχο κανόνα ΠΣΑ **type(type(“place”)).** Αφού δημιουργηθούν οι τύποι, δημιουργούνται οι κανόνες οι οποίοι συσχετίζουν τις καταστάσεις των αντικειμένων με τους ανάλογους τύπους που τους αντιστοιχούν. Οι ακόλουθοι κανόνες θα δημιουργηθούν :

1) **variable(variable(("connected", X1, X2))) :- has(X1, type("place")), has(X2, type("place")).**
2) **variable(variable(("at-robot", X1))) :- has(X1, type("place")).**
3) **variable(variable(("visited", X1))) :- has(X1, type("place")).**

Έτσι γίνεται η συσχέτιση των αντικειμένων (**X1**, **X2**) με τον τύπο τους (**place**). Μέσο αυτής της μεθόδου γνωρίζει ο επιλυτής ποιος τύπος αντικειμένου μπορεί να χρησιμοποιηθεί σε κάθε περίπτωση. Για παράδειγμα ο πρώτος κανόνας δείχνει πως το κατηγόρημα **connected** αναφέρεται σε δύο αντικείμενα **X1** και **X2** τα οποία είναι και τα δύο τύπου **place** μέσω του κατηγορήματος **has(X1, type("place")), has(X2, type("place")).** Έπειτα δημιουργούνται οι κανόνες που αναφέρονται στις δράσεις. Η υλοποίηση των δράσεων αποτελεί πιο περίπλοκη διαδικασία αφού για όλες τις προϋποθέσεις (preconditions) και όλα τα αποτελέσματα των δράσεων (effect / postcondition) πρέπει να δημιουργηθεί διαφορετικός κανόνας.

Οι πιο κάτω κανόνες αποτελούν τη μετάφραση που επιστρέφει ο plasp όταν του δώσουμε το domain του Σχήματος 1.

1) **action(action(("move", X1, X2))) :- has(X1, type("place")), has(X2, type("place")).**
2) **precondition(action(("move", X1, X2)), variable(("at-robot", X1)), value(variable(("at-robot", X1)), true)) :- action(action(("move", X1, X2))).**
3) **precondition(action(("move", X1, X2)), variable(("connected", X1, X2)), value(variable(("connected", X1, X2)), true)) :- action(action(("move", X1, X2))).**
4) **postcondition(action(("move", X1, X2)), effect(unconditional), variable(("at-robot", X2)), value(variable(("at-robot", X2)), true)) :- action(action(("move", X1, X2))).**
5) **postcondition(action(("move", X1, X2)), effect(unconditional), variable(("at-robot", X1)), value(variable(("at-robot", X1)), false)) :- action(action(("move", X1, X2))).**
6) **postcondition(action(("move", X1, X2)), effect(unconditional), variable(("visited", X2)), value(variable(("visited", X2)), true)) :- action(action(("move", X1, X2))).**

Ο πρώτος κανόνας δείχνει ότι η δράση **move** χρειάζεται δυο αντικείμενα **X1** και **X2** τύπου **place** για να εκτελεστεί . Για να εκτελεστεί μια δράση πρέπει να πληρείται ο πρώτος κανόνας, να υπάρχουν δηλαδή τα αντικείμενα στο κόσμο και να έχουν το σωστό τύπο, και όλες οι προϋποθέσεις της δράσης. Ο δεύτερος κανόνας εκτελεί τον έλεγχο ότι ο πράκτορας βρίσκεται στη τοποθεσία **X1** ενώ ο τρίτος κανόνας ελέγχει εάν οι τοποθεσίες **X1** και **X2** είναι ενωμένες (**connected**) . Εφόσον πληρούνται αυτά τα κριτήρια η δράση **move** μπορεί να εκτελεστεί. Μόλις εκτελεστεί αυτή η δράση θα επιβληθούν στο κόσμο τα αποτελέσματα της πράξης. Ο τέταρτος κανόνας θέτει αληθές τη μεταβλητή **"at-robot", X2** δείχνοντας έτσι τη μετακίνηση του πράκτορα στη τοποθεσία **X2** ενώ ο πέμπτος κανόνας θέτει ψευδές τη μεταβλητή **"at-robot", X1** δείχνοντας έτσι πως ο πράκτορας δε βρίσκεται πλέον στη τοποθεσία **X1** . Τέλος ο έχτος κανόνας θέτει αληθές τη μεταβλητή **"visited", X2** δηλώνοντας έτσι ότι ο πράκτορας έχει επισκεφθεί τη τοποθεσία **X2**.
Αφού τελειώσει η μετάφραση του domain ξεκινάει η μετάφραση του problem. Το πρώτο βήμα του μεταφραστή είναι να μεταφράσει τα αντικείμενα που υπάρχουν στον κόσμο και να τους δώσει τον κατάλληλο τύπο. Οι κανόνες του Σχήματος 2 μεταφράζονται ως εξής :

1) **constant(constant("loc-x0-y0")).**
2) **has(constant("loc-x0-y0"), type("place")).**
3) **constant(constant("loc-x0-y1")).**
4) **has(constant("loc-x0-y1"), type("place")).**

5) **constant(constant("loc-x1-y0")).**

6) **has(constant("loc-x1-y0"), type("place")).**
7) **constant(constant("loc-x1-y1")).**
8) **has(constant("loc-x1-y1"), type("place")).**

Οι κανόνες 1,3,5 και 7 δηλώνουν τα αντικείμενα που υπάρχουν στο κόσμο ενώ οι κανόνες 2,4,6 και 8 το τύπο που αντιστοιχεί σε κάθε αντικείμενο. Δημιουργήθηκαν δηλαδή τέσσερεις τοποθεσίες οι οποίες είναι όλες τύπου **place**. Το επόμενο βήμα του μεταφραστή είναι η δημιουργία της αρχικής κατάστασης που επικρατεί στον κόσμο. Για τη δημιουργία της αρχικής κατάστασης χρησιμοποιείται το κατηγόρημα **initialState** μέσα στο οποίο αποθηκεύονται όλες οι καταστάσεις στις οποίες βρίσκονται τα αντικείμενα του κόσμου. Η αρχική κατάσταση ορίζεται ως εξής :

1) **initialState(variable(("at-robot", constant("loc-x1-y1"))), value(variable(("at-robot", constant("loc-x1-y1"))), true)).**
2) **initialState(variable(("visited", constant("loc-x1-y1"))), value(variable(("visited", constant("loc-x1-y1"))), true)).**
3) **initialState(variable(("connected", constant("loc-x0-y0"), constant("loc-x1-y0"))), value(variable(("connected", constant("loc-x0-y0"), constant("loc-x1-y0"))), true)).**
4) **initialState(variable(("connected", constant("loc-x0-y0"), constant("loc-x0-y1"))), value(variable(("connected", constant("loc-x0-y0"), constant("loc-x0-y1"))), true)).**
5) **initialState(variable(("connected", constant("loc-x0-y1"), constant("loc-x1-y1"))), value(variable(("connected", constant("loc-x0-y1"), constant("loc-x1-y1"))), true)).**
6) **initialState(variable(("connected", constant("loc-x0-y1"), constant("loc-x0-y0"))), value(variable(("connected", constant("loc-x0-y1"), constant("loc-x0-y0"))), true)).**
7) **initialState(variable(("connected", constant("loc-x1-y0"), constant("loc-x0-y0"))), value(variable(("connected", constant("loc-x1-y0"), constant("loc-x0-y0"))), true)).**
8) **initialState(variable(("connected", constant("loc-x1-y0"), constant("loc-x1-y1"))), value(variable(("connected", constant("loc-x1-y0"), constant("loc-x1-y1"))), true)).**
9) **initialState(variable(("connected", constant("loc-x1-y1"), constant("loc-x0-y1"))), value(variable(("connected", constant("loc-x1-y1"), constant("loc-x0-y1"))), true)).**
10) **initialState(variable(("connected", constant("loc-x1-y1"), constant("loc-x1-y0"))), value(variable(("connected", constant("loc-x1-y1"), constant("loc-x1-y0"))), true)).**
11) **initialState(X, value(X, false)) :- variable(X), not initialState(X, value(X, true)).**

Ο πρώτος κανόνας υποδηλώνει τη θέση στην οποία βρίσκεται ο πράκτορας στο στιγμιότυπο του προβλήματος. Ο δεύτερος κανόνας θέτει τη μεταβλητή που υποδηλώνει πως ο πράκτορας

επισκέφτηκε την αρχική τοποθεσία σε αληθές. Οι κανόνες 3 μέχρι 10 δημιουργούν σταθερές οι οποίες δείχνουν ποιες τοποθεσίες είναι ενωμένες μεταξύ τους. Ο τελευταίος κανόνας θέτει τις υπόλοιπες καταστάσεις που δεν υπάρχουν στον κόσμο ως ψευδές . Για παράδειγμα για τη κατάσταση “at-robot” θα δημιουργηθούν οι αντίστοιχοι κανόνες

Στο initial state του μεταφρασμένου παραδείγματος υπάρχουν οι κανόνες

initialState(variable(("at-robot", constant("loc-x0-y1"))), value(variable(("at-robot", constant("loc-x1-y1"))), false)).

initialState(variable(("at-robot", constant("loc-x0-y0"))), value(variable(("at-robot", constant("loc-x1-y1"))), false)).

initialState(variable(("at-robot", constant("loc-x1-y0"))), value(variable(("at-robot", constant("loc-x1-y1"))), false)).

Το τελευταίο βήμα του μεταφραστή για την ολοκλήρωση της μετατροπής σε ΠΣΑ πρόγραμμα είναι να δημιουργήσει τις καταστάσεις στόχων του προβλήματος. Οι πιο κάτω κανόνες είναι η μετάφραση που επιστρέφει ο plasp για τους στόχους του προβλήματος. Οι τέσσερεις κανόνες δηλώνουν πως ο πράκτορας πρέπει να επισκεφθεί όλες τις τοποθεσίες του πλέγματος έτσι ώστε να επιλυθεί το πρόβλημα. Με τη μετάφραση των στόχων ο επιλυτής τελειώνει την εκτέλεση του δημιουργώντας όλους τους κανόνες που αντιστοιχούν στο αρχικό πρόβλημα σε ΠΣΑ μορφή.

1) **goal(variable(("visited", constant("loc-x0-y0"))), value(variable(("visited", constant("loc-x0-y0"))), true)).**
2) **goal(variable(("visited", constant("loc-x0-y1"))), value(variable(("visited", constant("loc-x0-y1"))), true)).**
3) **goal(variable(("visited", constant("loc-x1-y0"))), value(variable(("visited", constant("loc-x1-y0"))), true)).**
4) **goal(variable(("visited", constant("loc-x1-y1"))), value(variable(("visited", constant("loc-x1-y1"))), true)).**

2.7 Πρόγραμμα επίλυσης ΠΠΔ

Για να μπορέσουμε να επιλύσουμε Προβλήματα Δράσης μέσω ΠΣΑ χρειαζόμαστε ένα πρόγραμμα ΠΣΑ το οποίο κωδικοποιεί τη σημασιολογία τέτοιου είδους προβλήματα. Υπάρχουν έτοιμα πρωτότυπα προγράμματα προβλημάτων, έτσι δε χρειάζεται να γίνει η υλοποίηση από την αρχή. Για την υλοποίηση των ευρετικών της εργασίας αυτής χρησιμοποιήθηκε το πρόγραμμα που θα αναλυθεί πιο κάτω. Μέσω του ΠΣΑ προγράμματος μας δίνεται η επιλογή εκτέλεσης σειριακών δράσεων και παράλληλων δράσεων. Αν επιλεχθούν να εκτελεστούν παράλληλες δράσεις τότε μπορούμε να επιλέξουμε επιπλέον χαλαρώσεις που μπορούν να υπάρξουν στη διαδικασία αναζήτησης. Συγκεκριμένα υπάρχουν 5 επιλογές για τη παράλληλη εκτέλεση δράσεων. Η πρώτη επιλογή αγνοεί τον αμοιβαίο αποκλεισμό που υπάρχει μεταξύ δύο δράσεων, αν δηλαδή δύο δράσεις έχουν περιορισμό να μην μπορούν να εκτελεστούν παράλληλα θα αγνοηθεί ο περιορισμός αυτός. Η δεύτερη επιλογή επιστρέφει δράσεις οι οποίες μπορούν να εκτελεστούν με οποιαδήποτε σειρά σε κάποια χρονική στιγμή, δηλαδή δεν έχει καμιά από τις δράσεις που εκτελούνται προϋποθέσεις από τα αποτελέσματα μιας άλλης δράσης που εκτελέστηκε την ίδια χρονική στιγμή. Η τρίτη επιλογή επιστρέφει δράσεις οι οποίες εκτελούνται με συγκεκριμένη σειρά , δηλαδή τα αποτελέσματα μιας από τις δράσεις που θα εκτελεστούν μπορεί να επηρεάζουν τη δυνατότητα εκτέλεσης μιας άλλης από τις δράσεις που επιλέχτηκαν την ίδια χρονική στιγμή. Η τέταρτη επιλογή επιτρέπει σε δράσεις να εκτελούνται παραλείποντας την επίτευξη των προϋποθέσεων τους. Η τελευταία επιλογή δουλεύει όπως και η τέταρτη με τη διαφορά ότι υπάρχει αποφυγή της κυκλικής επανάληψης στις δράσεις που επιλέγονται. Η επιλογή για την εκτέλεση πράξεων μπορεί να περαστεί ως παράμετρος στον επιλυτή ή να τοποθετηθεί απευθείας στο πρόγραμμα μέσω της σταθεράς **#const _parallel = X.** όπου X η επιλογή από τις πιο πάνω περιγραφές αρχίζοντας με την πρώτη επιλογή για X = 0 μέχρι την τελευταία επιλογή με X = 4. Για οποιοδήποτε άλλο X εκτελείται η σειριακή διαδικασία. Για την εκτέλεση των πειραμάτων χρησιμοποιήθηκε **#const _parallel = 1.** εκτελούνται δηλαδή παράλληλες πράξεις οι οποίες δεν επηρεάζουν τις προϋποθέσεις άλλων πράξεων που εκτελούνται την ίδια χρονική στιγμή. Υπάρχει ακόμη η δυνατότητα να μην εκτελεστεί δράση σε κάποια χρονική στιγμή. Η σταθερά **#const planner_on = X.** για X = 0 δεν επιτρέπει την αδράνεια, πρέπει δηλαδή να εκτελεστεί τουλάχιστον μια δράση ανά χρονική στιγμή, ενώ για X != 0 μπορούν να μην εκτελεστούν δράσεις σε κάποιες χρονικές στιγμές. Για την εκτέλεση των πειραμάτων δεν χρησιμοποιήθηκαν αδρανείς χρονικές στιγμές.

Για την ανάλυση των κανόνων του προγράμματος θα χρησιμοποιήσουμε ένα απλό παράδειγμα κόσμου. Έστω ότι έχουμε ένα πλέγμα (n x n) και κάθε τοποθεσία του πλέγματος είναι ένα αντικείμενο place του κόσμου. Στον κόσμο υπάρχει ένας πράκτορας – ρομπότ ο οποίος εξερευνά τις τοποθεσίες του κόσμου. Η κατάσταση κάθε τοποθεσίας που εξερευνάει ο πράκτορας θεωρείται ως visited. Η κατάσταση στην οποία βρίσκεται ο πράκτορας, δηλαδή η τοποθεσία που βρίσκεται κάποια χρονική στιγμή, καθορίζεται μέσω της κατάστασης at-robot. Δύο γειτονικές τοποθεσίες θεωρούνται ενωμένες αν υπάρχει κατάσταση connected για τις τοποθεσίες αυτές. Ο πράκτορας μπορεί να μετακινηθεί από την παρούσα του τοποθεσία σε μια άλλη μόνο αν είναι ενωμένες. Στόχος του πράκτορα είναι να εξερευνήσει όλες τις τοποθεσίες.
Αφού ορίσαμε τον κόσμο ας δούμε πως

fluent(X): Το κατηγόρημα αυτό περιέχει μια κατάσταση (X) στην οποία μπορεί να βρίσκεται κάποιο αντικείμενο του κόσμου. Για το κόσμο που θέσαμε θα δημιουργηθεί ένα κατηγόρημα fluent για κάθε πιθανή θέση που βρίσκεται ο πράκτορας fluent(variable("at-robot", X)), όπου X μια τοποθεσία του κόσμου, για κάθε ενωμένο ζεύγος τοποθεσιών fluent(variable("connected",X,V)) ,όπου X και V τοποθεσίες του κόσμου, και για κάθε τοποθεσία που μπορεί να επισκεφθεί ο πράκτορας fluent(variable("visited",X)) όπου X μια τοποθεσία του κόσμου.

fluent (X, V): Το κατηγόρημα αυτό περιέχει μια κατάσταση (X) στην οποία μπορεί να βρίσκεται κάποιο αντικείμενο του κόσμου καθώς και τις τιμές αληθείας της κατάστασης (V), αν δηλαδή ισχύει ή όχι στον κόσμο. Δημιουργούνται κατηγορήματα όπως και στο fluent(X) αλλά αυτή τη φορά υπάρχουν και όλες οι πιθανές τιμές αληθείας που μπορεί να πάρει η κατάσταση στον κόσμο. Για παράδειγμα , το κατηγόρημα fluent(variable("visited",X), value(variable("visited",X),true)) και fluent(variable("visited",X), value(variable("visited",X),false)) θα δημιουργηθούν για κάθε τοποθεσία του κόσμου εχτός από την αρχική τοποθεσία που βρίσκεται ο πράκτορας. Ο λόγος που συμβαίνει αυτό είναι ότι κάθε τοποθεσία εκτός της αρχικής αρχίζει στον κόσμο με τιμή αληθείας false αλλά μπορεί να μετατραπεί σε true όταν ο πράκτορας την εξερευνήσει. Για την αρχική τοποθεσία όμως δεν ισχύει αυτό αφού μπορεί να έχει μόνο τιμή αληθείας true, δεν μπορεί δηλαδή να περάσει από κατάσταση visited σε κατάσταση not visited μέσω κάποιας δράσης. Για το κατηγόρημα connected υπάρχουν μόνο τιμές αληθείας true αφού δεν μπορούν να μεταβληθούν οι καταστάσεις των ενώσεων του κόσμου και για το κατηγόρημα at-robot υπάρχουν true και false αφού ο πράκτορας μπορεί να περάσει από κάποια τοποθεσία όσες φορές χρειαστεί έτσι ώστε να εξερευνηθεί το πλέγμα.

holds (X, V, t): Το κατηγόρημα αυτό περιέχει μια κατάσταση (X) στην οποία μπορεί να βρίσκεται κάποιο αντικείμενο του κόσμου καθώς και την τιμή αληθείας της κατάστασης (V) για μια χρονική στιγμή t. Στο holds (X, V, t) αποθηκεύεται η κατάσταση στην οποία βρίσκεται ο κόσμος κάθε χρονική στιγμή. Η διαφορά του κατηγορήματος holds από το fluent είναι ότι περιέχει μόνο μια τιμή αληθείας για κάθε κατάσταση. Για παράδειγμα το κατηγόρημα holds(variable("visited",X), value(variable("visited",X),Z),t) όπου X η τοποθεσία, Z η τιμή αληθείας και t η χρονική στιγμή, μπορεί να δώσει τιμή true ή false στη μεταβλητή Z, δεν μπορεί δηλαδή να φυλάξει και τις δύο τιμές αληθείας σε μια χρονική στιγμή, .

occurs (A, t): Το κατηγόρημα αυτό περιέχει μια δράση (A) και μια χρονική στιγμή t. Στο occurs (A, t) αποθηκεύονται όλες οι δράσεις που εκτελούνται κάθε χρονική στιγμή καθώς και η χρονική στιγμή στην οποία εκτελέστηκαν. Για παράδειγμα το κατηγόρημα occurs(action("move",X,V),t) όπου X η τοποθεσία που βρίσκεται ο πράκτορας, V η τοποθεσία στην οποία θα μετακινηθεί και t η χρονική στιγμή στην οποία εκτελείται αυτή η δράση, είναι η δήλωση μιας μετακίνησης του πράκτορα σε μια τοποθεσία του πλέγματος.

precondition (A, X, V): Το κατηγόρημα αυτό περιέχει μια κατάσταση (X) στην οποία πρέπει να βρίσκεται κάποιο αντικείμενο του κόσμου καθώς και την τιμή αληθείας που πρέπει να έχει η κατάσταση (V) έτσι ώστε να μπορεί να εκτελεστεί μια δράση (A). Στο precondition(A,X,V) αποθηκεύονται στην προ επεξεργασία των δεδομένων (Κεφάλαιο 2.6) όλες οι προϋποθέσεις που έχει η κάθε δράση έτσι ώστε να μπορεί να εκτελεστεί. Τα κατηγορήματα precondition που θα δημιουργηθούν για κάθε πιθανή κίνηση που μπορεί να εκτελεστεί στον κόσμο είναι τα εξής:

precondition (action ("move", X, V), variable ("at-robot", X), value (variable ("at-robot", X), true))
precondition (action ("move", X, V), variable ("connected", X, V), value (variable ("connected", X, V), true))

Το X χαρακτηρίζει την τοποθεσία που βρίσκεται ο πράκτορας κάποια χρονική στιγμή και το V τη νέα τοποθεσία στην οποία θα μεταβεί. Ο πρώτος κανόνας ελέγχει ότι ο πράκτορας βρίσκεται στη τοποθεσία X ενώ ο δεύτερος ελέγχει ότι οι τοποθεσίες X και V είναι ενωμένες (connected) έτσι ώστε να μπορέσει να εκτελεστεί η δράση.

postcondition (A, X, V): Το κατηγόρημα αυτό περιέχει μια κατάσταση (X) στην οποία θα βρεθεί κάποιο αντικείμενο του κόσμου καθώς και την τιμή αληθείας που θα πάρει η κατάσταση (V) εάν εκτελεστεί μια δράση (A). Στο postcondition (A, X, V) αποθηκεύονται στην προ επεξεργασία των δεδομένων (Κεφάλαιο 2.6) όλες οι αλλαγές (effects) που θα συμβούν στον κόσμο όταν εκτελεστεί

οποιαδήποτε δράση. Τα κατηγορήματα postcondition που θα δημιουργηθούν για κάθε πιθανή κίνηση που μπορεί να εκτελεστεί στον κόσμο είναι τα εξής:

postcondition (action ("move", X, V), variable ("at-robot", V), value (variable ("at-robot", V), true))
postcondition (action ("move", X, V), variable ("at-robot", X), value (variable ("at-robot", X), false))
postcondition (action ("move", X, V), variable ("visited", V), value (variable ("visited", V), true))

Το X χαρακτηρίζει την τοποθεσία που βρίσκεται ο πράκτορας πριν την εκτέλεση της δράσης και το V τη νέα τοποθεσία στην οποία θα μεταβεί. Ο πρώτος κανόνας θέτει τον πράκτορα στη νέα τοποθεσία. Ο δεύτερος κανόνας θέτει τη τιμή αληθείας της αρχικής τοποθεσίας σε false δείχνοντας έτσι ότι δε βρίσκεται πλέον σε αυτή τη τοποθεσία. Ο τρίτος κανόνας θέτει την τιμή αληθείας της κατάστασης visited για την νέα τοποθεσία σε true αφού ο πράκτορας έχει πλέον επισκεφθεί αυτή τη θέση.

single (X, t): Το κατηγόρημα αυτό αναφέρεται σε μια κατάσταση (X) στην οποία βρίσκεται κάποιο αντικείμενο του κόσμου κάποια χρονική στιγμή t. Το κατηγόρημα single (X t,) χρησιμοποιείται σε περιορισμούς έτσι ώστε να αποτρέψει την παράλληλη επεξεργασία ενός ατόμου – αντικειμένου από πολλές δράσεις ταυτόχρονα. Στο κόσμο που ορίσαμε η μόνη κατάσταση που μπορεί να επηρεαστεί από διάφορες κινήσεις ανά πάσα χρονική στιγμή είναι η τοποθεσία του πράκτορα. Έτσι σε κάθε χρονική στιγμή θα δημιουργείται ένα κατηγόρημα single(variable("at-robot", X),t), όπου X η τοποθεσία στην οποία βρίσκεται ο πράκτορας τη χρονική στιγμή t, έτσι ώστε να μπορεί μόνο μία δράση να επηρεάσει το άτομο αυτό.

Αφού αναλύσαμε τα κύρια κατηγορήματα που χρησιμοποιούνται για την αναπαράσταση του κόσμου απομένει να αναλυθούν οι κανόνες του προγράμματος ΠΣΑ.

selfdefeat(A,X) :- active(A), precondition(A,X,V), _parallel = 1, has_condition(A,X,1), not postcondition(A,X,V).

Ο κανόνας δημιουργεί κατηγόρημα selfdefeat (A, X), όπου A μια δράση που μπορεί να εκτελεστεί στον κόσμο και X μια κατάσταση του κόσμου, όταν η τιμή αληθείας της κατάστασης στον κόσμο αλλάζει με την εκτέλεση της δράσης A. Αν δηλαδή η κατάσταση X στο precondition είχε τιμή αληθείας true τότε για να δημιουργηθεί το κατηγόρημα selfdefeat πρέπει στο postcondition η

κατάσταση X να έχει τιμή αληθείας false , ή και το αντίστροφο (precondition false / postcondition true). Το κατηγόρημα αυτό χρησιμοποιείται αργότερα για να αποκλειστούν εκτελέσεις κινήσεων σε ένα άτομο από πολλαπλές κινήσεις.

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

Μέσω των πιο πάνω κανόνων δημιουργούνται όλες οι πιθανές καταστάσεις που μπορούν να υπάρξουν στον κόσμο. Το κατηγόρημα produce (X, V) αναφέρεται σε καταστάσεις οι οποίες μπορούν να αλλάξουν τιμή αληθείας κατά την επίλυση του προβλήματος, για παράδειγμα η κατάσταση "at-robot" και η κατάσταση "visited" βρίσκονται στην κατηγορία αυτή. Το κατηγόρημα persist (X, V) αναφέρεται σε καταστάσεις οι οποίες έχουν σταθερή τιμή αληθείας στον κόσμο, για παράδειγμα η κατάσταση "connected" έχει πάντα τιμή αληθείας true και δεν μεταβάλλεται κατά την επίλυση. Ο 3ος κανόνας δημιουργεί fluent με βάση την αρχική κατάσταση του κόσμου και ο 4ος με βάση τις μεταβολές των καταστάσεων που επιφέρουν οι δράσεις στον κόσμο. Ο 5ος κανόνας δημιουργεί κατηγορήματα τύπου fluent(X) για κάθε fluent (X, V) που υπάρχει. Το κατηγόρημα fluent(X) χρησιμοποιείται σε κάποιους κανόνες αντί του κατηγορήματος fluent (X, V) διότι χρειάζεται η συσχέτιση μιας κατάστασης (X) μεταξύ διαφόρων κατηγορημάτων αλλά χωρίς να μας ενδιαφέρει η τιμή αληθείας τους (V) στον κόσμο.

holds(X,V,0) :- initialState(X,V), fluent(X).

Μέσω του πιο πάνω κανόνα δημιουργείται η αρχική κατάσταση του κόσμου η οποία αποθηκεύεται στο κατηγόρημα **holds(X,V,t),** όπου X η κατάσταση στην οποία βρίσκεται κάποιο αντικείμενο, V η τιμή αληθείας της στον κόσμο και t η χρονική στιγμή στην οποία ισχύει η κατάσταση αυτή. Το κατηγόρημα holds κρατάει την κατάσταση του κόσμου για κάθε χρονική στιγμή. Η αρχική κατάσταση αποθηκεύεται στη χρονική στιγμή 0.

:- fluent(X), #count{V : holds(X,V,0)} > 1.

Ο πιο πάνω περιορισμός απαγορεύει σε οποιοδήποτε αντικείμενο υπάρχει στην αρχική κατάσταση του κόσμου να έχει περισσότερες από μία καταστάσεις. Δεν μπορεί δηλαδή να υπάρχει το

κατηγόρημα holds(variable("visited",X), value(variable("visited",X),true),0) και το κατηγόρημα holds(variable("visited",X), value(variable("visited",X),false),0) στην αρχική κατάσταση του κόσμου, δημιουργείται αντίφαση.

1 {holds(X,V,t) : fluent(X,V)} 1 :- fluent(X).

Ο πιο πάνω περιορισμός απαγορεύει σε οποιοδήποτε κατάσταση υπάρχει στον κόσμου να έχει περισσότερες από μία καταστάσεις. Λειτουργεί δηλαδή όπως και ο προηγούμενος κανόνας με διαφορά ότι χρησιμοποιείται για όλες χρονικές στιγμές που θα υπάρξουν και είναι μέρος της επαναληπτικής διαδικασίας, ενώ ο προηγούμενος κανόνας αφορά μόνο την αρχική κατάσταση του κόσμου.

:- planner_on = 0, not occurs(A,t) : active(A).

Ο πιο πάνω περιορισμός αποτρέπει τις αδρανείς κινήσεις στον κόσμο αν η σταθερά planner_on έχει τεθεί στην τιμή 0. Δεν μπορεί δηλαδή σε κάποια χρονική στιγμή να μην εκτελεστεί κάποια δράση.

:- occurs(A,t), postcondition(A,X,V), fluent(X), not holds(X,V,t).

Ο πιο πάνω περιορισμός ελέγχει ότι κάθε δράση που εκτελέστηκε στον κόσμο έχει επιφέρει τα αποτελέσματα που έπρεπε στον κόσμο. Δεν επιτρέπει δηλαδή να εκτελεστεί μια δράση και η τιμή αληθείας που έπρεπε να έχει επιφέρει σε μια κατάσταση του κόσμου να μην υπάρχει στο κατηγόρημα holds (X, V ,t)

:- _inertia = 0, holds(X,V,t), not holds(X,V,t-1),
not occurs(A,t) : active(A), postcondition(A,X,V), not precondition(A,X,V).

Ο πιο πάνω περιορισμός επιβεβαιώνει ότι για κάθε κατάσταση που βρίσκεται στο κόσμο μια χρονική στιγμή t και δεν υπήρχε την προηγούμενη χρονική στιγμή, εκτελέστηκε μια δράση που επίφερε τα αποτελέσματα που υπάρχουν στον κόσμο τη παρούσα χρονική στιγμή. Επιβεβαιώνει δηλαδή ότι κάθε αλλαγή που έγινε στον κόσμο έχει γίνει μέσω των αποτελεσμάτων μιας δράσης.

:- _parallel != 0, _parallel != 1, _parallel != 2, _parallel != 3, _parallel != 4,
#count{A : occurs(A,t)} > 1.

Ο πιο πάνω περιορισμός αποτρέπει την εκτέλεση πολλαπλών δράσεων κατά τη σειριακή διαδικασία.

:- _parallel = 1, occurs(A,t), precondition(A,X,V), not has_condition(A,X,1), not holds(X,V,t).

Ο πιο πάνω περιορισμός ελέγχει ότι οι προϋποθέσεις μιας κίνησης που έχει εκτελεστεί πληρούνται, δηλαδή ότι οι προϋποθέσεις για κάθε κατάσταση που περνά στο κατηγόρημα holds μια χρονική στιγμή μέσω της εκτέλεσης μιας δράσης A είναι καλύφθηκαν.

single(X,t) :- occurs(A,t), selfdefeat(A,X).

Ο πιο πάνω κανόνας δημιουργεί κατηγόρημα single (X, t) για κάθε κατάσταση που αλλάζει μέσω μιας δράσης τη χρονική στιγμή t.

:- single(X,t), #count{A : occurs(A,t), postcondition(A,X,V), not precondition(A,X,V)} > 1.

Ο πιο πάνω περιορισμός αποτρέπει την παράλληλη επεξεργασία μιας κατάστασης του κόσμου από πολλαπλές δράσης την ίδια χρονική στιγμή. Αν δηλαδή δύο δράσεις επηρεάζουν μια από τις καταστάσεις του κόσμου, τότε δεν θα επιτραπεί μέσω του περιορισμού αυτού να εκτελεστούν και οι δύο δράσεις την ίδια χρονική στιγμή.

:- query(t), goal(X,V), not holds(X,V,t).

Μέσω του πιο πάνω κανόνα γίνεται ο έλεγχος αν η κατάσταση στην οποία βρίσκεται ο κόσμος κάποια χρονική στιγμή t αποτελεί επίτευξη των στόχων του προβλήματος. Ελέγχει δηλαδή αν στο κατηγόρημα holds υπάρχουν όλες οι καταστάσεις που υπάρχουν στο κατηγόρημα goal, το όποιο περιέχει την τελική κατάσταση στην οποία πρέπει να βρίσκεται ο κόσμος για να μπορεί να τερματιστεί η επίλυση.

#show occurs/2.

Ο πιο πάνω κανόνας επιστρέφει στην έξοδο του προγράμματος όλες τις κινήσεις που εκτελέστηκαν κατά τη διαδικασία επίλυσης.

Κεφάλαιο 3

Ευρετικές Μέθοδοί Στην Επίλυση ΠΠΔ Με ΠΣΑ

3.1 Ευρετικές Μέθοδοι

Μια ευρετική μέθοδος είναι μια διαφορετική προσέγγιση σε κάποιο πρόβλημα. Η ευρετική μέθοδος μπορεί να επιφέρει καλύτερα αποτελέσματα από την κωδικοποίηση χωρίς ευρετικό στις πλείστες περιπτώσεις αλλά δεν είναι εγγυημένο ότι η λύση που θα δοθεί θα είναι η καλύτερη ή η πιο αποδοτική. Σε προβλήματα εξερεύνησης τα ευρετικά δεν χρησιμοποιούνται για να βρίσκουμε την καλύτερη λύση αλλά για να βρίσκουμε ποιο γρήγορα μία λύση, μειώνοντας το χώρο αναζήτησης λύσεων. Πολλές φορές μπορούν να χρησιμοποιηθούν για να φτάσουμε σε ένα μέσο σημείο του πλάνου και έπειτα στη τελική λύση με πιο γρήγορο τρόπο. Γενικά επηρεάζουν το τρόπο με τον οποίο ο έξυπνος πράκτορας θα εκτελεί τις κινήσεις του, είτε μέσο περιορισμών είτε μέσο μεγιστοποίησης ή ελαχιστοποίησης τιμών. Τα ευρετικά δημιουργούνται μέσω εμπειριών που έχουμε από προβλήματα που αντιμετωπίσαμε προηγουμένως. Αν για παράδειγμα σε κάποιο πρόβλημα εντοπίζαμε μείωση απόδοσης λόγο αυξημένης εξερεύνησης τότε μέσω ενός ευρετικού μπορεί να περιοριστεί ο αριθμός των βημάτων εξερεύνησης και να βελτιωθεί η ταχύτητα. Σε ένα νέο πρόβλημα που εμφανίζει παρόμοια συμπεριφορά θα εφαρμόσουμε παρόμοια στρατηγική έτσι ώστε να έχουμε καλύτερα αποτελέσματα. Για να βεβαιωθούμε ότι ένα ευρετικό δουλεύει σωστά πρέπει να γίνει πειραματική ανάλυση με ένα δείγμα προβλημάτων έτσι ώστε να είμαστε σίγουροι πως επιφέρει θετικά αποτελέσματα.

3.2 Χρήση Ευρετικών Μέθοδων για επίλυση ΠΠΔ μέσω ΠΣΑ

Οι περισσότεροι επιλυτές ΠΣΑ και SAT λειτουργούν με μια στρατηγική σταθερού μήκους πλάνου. Ψάχνουν δηλαδή να βρουν τη λύση χρησιμοποιώντας σειριακούς αλγορίθμους αρχίζοντας από τη χρονική στιγμή 0 και αυξάνουν το μήκος του πλάνου κατά 1 μέχρι να βρεθεί το πλάνο που επιλύει το πρόβλημα. Αν θα χρησιμοποιηθούν παράλληλες πράξεις τότε ο χρόνος εκτέλεσης μπορεί να μειωθεί με τη χρήση κάποιων τεχνικών. Μια από αυτές τις τεχνικές εκτελεί παράλληλα n πλάνα μέχρι να βρεθεί η λύση. Αν δηλαδή το n = 4 τότε θα ξεκινήσει να εκτελεί τα πλάνα με μήκος 0-3 και μόλις τελειώνει ένα πλάνο θα προστίθεται στα πλάνα που εκτελούνται το επόμενο πλάνο στη σειρά μέχρι να βρεθεί η λύση. Όταν δηλαδή περάσουν 2 μονάδες χρόνου θα έχουν τελειώσει τα πλάνα με μήκος 0 και 1 και θα προστεθούν για επίλυση τα πλάνα με μήκος 4 και 5. Η διαδικασία θα επαναλαμβάνεται μέχρι να βρεθεί το πλάνο που βρίσκει τη λύση [2]. Αυτό μπορεί να επιφέρει αρκετή εξοικονόμηση χρόνου ανάλογα με την τιμή του n και τη χρονική στιγμή στην οποία βρίσκεται το πλάνο που λύνει το πρόβλημα. Η εξοικονόμηση χρόνου εδώ γίνεται μέσω του επιλυτή, δηλαδή μέσω μεθόδων που βελτιστοποιούν την ταχύτητα του επιλυτή ανεξάρτητα με τι θα δοθεί σαν είσοδος για επίλυση. Για την περεταίρω μείωση του χρόνου εκτέλεσης μπορούν να χρησιμοποιηθούν ευρετικά τα οποία προσπαθούν να βελτιστοποιήσουν το τρόπος αναζήτησης του επιλυτή. Αν έχουμε για παράδειγμα ένα πρόβλημα στο οποίο πρέπει να βρεθεί κάποια διαδρομή, αν σε κάποια από τα μονοπάτια υπάρχει ποινή αν τα επισκεφτεί ο επιλυτής τότε μπορεί να δημιουργηθεί ένα ευρετικό το οποίο θα προσπαθήσει να βρει τη λύση αποφεύγοντας τα μονοπάτια με τις ποινές όσο το δυνατό περισσότερο. Έτσι με τη χρήση του ευρετικού αλλάζει η αρχική λύση που θα παρήγαγε ο επιλυτής σε μια λύση που λαμβάνει υπόψη επιπλέον κανόνες επίλυσης . Όπως αναφέρθηκε και πριν στόχος του ευρετικού δεν είναι η εύρεση τη βέλτιστης λύσης αλλά ενός πιο σύντομου μονοπατιού. Έτσι το ευρετικό μπορεί να χρησιμοποιηθεί σε διάφορα προβλήματα χωρίς να αλλαχτεί. Ο επιλυτής clingo έχει τη δυνατότητα χρήσης ευρετικών με τη χρήση του κανόνα ΠΣΑ #heuristic. Μπορούν όμως να δημιουργηθούν ευρετικά χωρίς να γίνουν αλλαγές στον κώδικα του επιλυτή. Για τη δημιουργία των δικών μας ευρετικών αποφασίσαμε πως δεν θα αλλαχτεί ο κώδικας του επιλυτή αλλά θα δημιουργηθεί ένα ευρετικό που εκμεταλλεύεται τις δυνατότητες του. Δημιουργήσαμε λοιπόν ένα ευρετικό το οποίο προσπαθεί να ελαχιστοποιήσει τις κινήσεις που απομένουν μέχρι να λυθεί το πρόβλημα και τρία ευρετικά τα οποία προσπαθούν να μεγιστοποιήσουν τους στόχους που επιτεύχθηκαν μέχρι την επίλυση του προβλήματος. Για να δουλέψουν τα ευρετικά πρέπει το πρόβλημα να σπάσει σε πιο μικρά υποπροβλήματα και να καλείται ο επιλυτής να λύσει το κάθε υποπρόβλημα επαναληπτικά μέχρι να βρεθεί η συνολική λύση. Έτσι με κάθε κλήση του επιλυτή γίνεται ελαχιστοποίηση των κινήσεων που απομένουν μέχρι τη λύση και μεγιστοποιήσει των στόχων που επιτεύχθηκαν. Αυτή

η μέθοδος μπορεί να επιφέρει επιτάχυνση στη λύση αλλά κυρίως αποφεύγει την εξερεύνηση μονοπατιών που δεν οδηγούν σε λύση.

3.3 Ανάλυση ευρετικών μεθόδων πειραματικής διαδικασίας

Για τη πειραματική διαδικασία δημιουργήθηκαν τέσσερεις ευρετικοί μέθοδοι. Το ευρετικό κινήσεων, το οποίο προσπαθεί να ελαχιστοποιήσει τις κινήσεις που απομένουν μέχρι τη λύση του προβλήματος και τα ευρετικά στόχων, καταστάσεων και συνδυασμού , τα οποία προσπαθούν να μεγιστοποιήσουν τους στόχους που έχουν επιτευχθεί μέχρι τη λύση. Τα τρία τελευταία λειτουργούν με την ίδια λογική αλλά έχουν διαφορετικό κατώφλι στο οποίο τερματίζεται η επαναληπτική διαδικασία επίλυσης. Για την δημιουργία των ευρετικών προστέθηκαν επιπρόσθετοι κανόνες ΠΣΑ στο πρόγραμμα που αναφέρθηκε στο Κεφάλαιο 2.7. Οι κανόνες που προστέθηκαν για την δημιουργία του ευρετικού κινήσεων είναι οι εξής :

```
newact(X,t):-occurs(X,t), t>c1.
#minimize{1,A,t:newact(A,t)}.
```

Ο πρώτος κανόνας δημιουργεί κατηγορήματα τύπου newact για κάθε κίνηση που εκτελείται μετά από την χρονική στιγμή c1. Η χρονική στιγμή c1 είναι η στιγμή που ξεκινάει ο επιλυτής να αγνοεί κάποιους από τους περιορισμούς που έχει έτσι ώστε να βρεθεί η λύση πιο σύντομα και να επιστραφεί μια πρόβλεψη για το πόσο κοντά είμαστε στη λύση του προβλήματος. Ο δεύτερος κανόνας προσπαθεί να ελαχιστοποιήσει των αριθμό των κατηγορημάτων newact δηλαδή ελαχιστοποιεί τον αριθμό των κινήσεων που θα εκτελεστούν μετά τα βήματα στα οποία υπάρχουν όλοι οι λογικοί περιορισμοί του προβλήματος. Με αυτό τον τρόπο επιστρέφεται η πρόβλεψη των απομενόντων κινήσεων μέχρι τη λύση του προβλήματος η οποία χρησιμοποιείται αργότερα στην επαναληπτική διαδικασία ως συνθήκη τερματισμού. Όταν ο αριθμός των απομενόντων κινήσεων είναι μικρότερος από 10 η επαναληπτική διαδικασία σταματάει.

Οι κανόνες που προστέθηκαν για την δημιουργία των υπολοίπων ευρετικών είναι οι εξής :

```
achnew(X,0) :- initialState(t,X,V),goal(X,V),t=c1.
achnew(X,K):- goal(X,V), holds(X,V,t),t=c1+K, num(K).
#minimize{-(24-3*K),X,K:achnew(X,K)}.
```

Ο πρώτος κανόνας δημιουργεί κατηγορήματα achnew για κάθε στόχο που έχει υλοποιηθεί στην αρχική κατάσταση του προβλήματος. Εφόσον το πρόβλημα χωρίζεται σε υποπροβλήματα σε κάθε

καινούργια αρχική κατάσταση που δίνεται για επίλυση θα υπάρχει διαφορετικός αριθμός επιλυμένων στόχων. Για τη διαδικασία αυτή οι στόχοι που έχουν επιτευχθεί σε αυτή τη φάση παίρνουν το μεγαλύτερο βάρος στη συνάρτηση του τρίτου κανόνα. Ο δεύτερος κανόνας δημιουργεί κατηγορήματα achnew για κάθε στόχο που έχει επιτευχθεί μετά τη χρονική στιγμή c1 μέχρι τη χρονική στιγμή c1+K όπου K ένας ακέραιος αριθμός, στη πειραματική διαδικασία έχει τιμή 8. Όσο πιο μεγάλο είναι το K τόσο πιο μικρό βάρος παίρνει στη συνάρτηση του τρίτου κανόνα. Ο τρίτος κανόνας υπολογίζει το συνολικό βάρος που έχουν οι στόχοι που έχουν επιτευχθεί μέχρι τη χρονική στιγμή c1+K και το επιστρέφει σαν μια εκτίμηση για το πόσο κοντά είμαστε στη λύση του προβλήματος. Έπειτα το κάθε ευρετικό με βάση το κατώφλι που έχει τερματίζει την επαναληπτική διαδικασία όταν ο αριθμός την πρόβλεψης αυτής υπερβεί το όριο. Το κατώφλι δημιουργείται στα τρία ευρετικά με τις εξής συναρτήσεις.

Ευρετικό στόχων: **threshold = goals * -100**
Ευρετικό καταστάσεων: **threshold = states * -25**
Ευρετικό συνδυασμού: **threshold = states * -12 - goals * 40**

Το ευρετικό στόχων εκτιμά το κατώφλι με βάση τον αριθμό των στόχων (μεταβλητή goals), οι οποίοι θα επιστραφούν μέσω της πρώτης κλίσης του ΠΣΑ προγράμματος, και το πολλαπλασιάζει με μια σταθερά (-100) . Το ευρετικό καταστάσεων εκτιμά το κατώφλι με βάση τον αριθμό των καταστάσεων που υπάρχουν μέσα στο PDDL problem file και το πολλαπλασιάζει με μια σταθερά (-25). Τέλος, το ευρετικό συνδυασμού χρησιμοποιεί τους στόχους και τον αριθμό των καταστάσεων για να εκτιμήσει το κατώφλι δίνοντας περίπου το μισό βάρος που είχε δοθεί στις προηγούμενες μεθόδους σε κάθε παράμετρο. Για τη σύγκριση της διαφοράς των ευρετικών όσο αφορά το κατώφλι θα αναλύσουμε τις τιμές των κατωφλίων που έχει το κάθε ένα από αυτά στο πιο κάτω PDDL πρόβλημα.

(define (problem grid-2)
(:domain grid-visit-all)
(:objects
loc-x0-y0
loc-x0-y1
loc-x1-y0
loc-x1-y1
- place)

(:init

(at-robot loc-x1-y1)

(visited loc-x1-y1)

(connected loc-x0-y0 loc-x1-y0)

(connected loc-x0-y0 loc-x0-y1)

(connected loc-x0-y1 loc-x1-y1)

(connected loc-x0-y1 loc-x0-y0)

(connected loc-x1-y0 loc-x0-y0)

(connected loc-x1-y0 loc-x1-y1)

(connected loc-x1-y1 loc-x0-y1)

(connected loc-x1-y1 loc-x1-y0))

(:goal

(and

(visited loc-x0-y0)

(visited loc-x0-y1)

(visited loc-x1-y0)

(visited loc-x1-y1))))

Όταν μεταφραστεί το πιο πάνω πρόβλημα θα επιστραφούν οι πιο στόχοι σε ΠΣΑ μορφή.

goal(variable(("visited", constant("loc-x0-y0"))), value(variable(("visited", constant("loc-x0-y0"))), true)).
goal(variable(("visited", constant("loc-x0-y1"))), value(variable(("visited", constant("loc-x0-y1"))), true)).
goal(variable(("visited", constant("loc-x1-y0"))), value(variable(("visited", constant("loc-x1-y0"))), true)).
goal(variable(("visited", constant("loc-x1-y1"))), value(variable(("visited", constant("loc-x1-y1"))), true)).

Το ευρετικό στόχων θα θέσει το κατώφλι στην τιμή -400 αφού υπάρχουν τέσσερεις στόχοι στο μεταφρασμένο πρόβλημα, το ευρετικό καταστάσεων θα θέσει το κατώφλι στην τιμή -250 αφού υπάρχουν 10 καταστάσεις στο αρχείο του PDDL προβλήματος και το ευρετικό συνδυασμού θα θέσει το κατώφλι στην τιμή -280 αφού υπάρχουν τέσσερεις στόχοι στο μεταφρασμένο πρόβλημα και 10 καταστάσεις στο αρχείο του PDDL προβλήματος. Φαίνεται λοιπόν για ένα μικρού μεγέθους πρόβλημα οι τιμές του κατωφλίου μπορούν να διαφέρουν αρκετά. Η διαφορά στον αριθμό του κατωφλίου παίζει σημαντικό ρόλο στο πόσο γρήγορα θα σταματήσει η επαναληπτική διαδικασία, με αποτέλεσμα κάποιες φορές να λύνεται το πρόβλημα σε λιγότερο χρόνο και άλλες φορές να αποτυγχάνει η διαδικασία επίλυσης διότι το κατώφλι ήταν πολύ μικρό και η επαναληπτική διαδικασία τέλειωσε πρόωρα.

Κεφάλαιο 4

Μεθοδολογία

4.1 Δημιουργία Προγράμματος

Με βάση όσα αναφέρθηκαν πιο πάνω θελήσαμε να δημιουργήσουμε αρχικά ένα πρόγραμμα σε Python μέσω του οποίου θα γίνονταν οι κλήσεις στον επιλυτή clingo. Το πρώτο βήμα είναι η κλήση του επιλυτή με το ΠΣΑ πρόγραμμα έτσι ώστε να πάρουμε μια πρόβλεψη για το πόσα βήματα θα χρειαστούν μέχρι να φτάσουμε στη λύση. Έπειτα καλείται ο επιλυτής ξανά αλλά αυτή τη φορά εκτελεί κ κανονικά βήματα. Αποθηκεύονται οι κινήσεις που εκτελέστηκαν καθώς και οι αλλαγές που επίφεραν στον κόσμο, δημιουργώντας έτσι ένα νέο αρχείο στο οποίο βρίσκεται η νέα αρχική κατάσταση του κόσμου και ένα αρχείο κινήσεων στο οποίο αποθηκεύονται όσες κινήσεις εκτελέστηκαν σε κάθε βήμα. Όταν τελειώσει η κλήση επιστρέφεται μια νέα εκτίμηση από τον επιλυτή η οποία μας δείχνει πόσο μακριά είναι η κατάσταση που βρισκόμαστε από τη λύση. Αν ο αριθμός αυτός είναι μεγαλύτερος του κατωφλίου που θέσαμε τότε καλούμε επαναληπτικά αυτή τη διαδικασία μέχρι να είναι μικρότερος του κατωφλίου. Μόλις περάσουμε τη τιμή του κατωφλίου καλούμε τον επιλυτή για τελευταία φορά αλλά με την κωδικοποίηση χωρίς ευρετικό έτσι ώστε να βρει τις απομένοντες κινήσεις που χρειάζονται μέχρι τη λύση. Όταν τελειώσει και η τελευταία κλήση αποθηκεύονται ξανά οι κινήσεις που εκτελέστηκαν στο αρχείο κινήσεων και γίνεται ο έλεγχος εγκυρότητας κινήσεων. Στο πρόβλημα εξερεύνησης του πράκτορα επάνω σε ένα 4x4 πλέγμα το κατώφλι ήταν -1600, -1250 και -1240 για τα ευρετικά στόχων, καταστάσεων και συνδυασμού αντίστοιχα. Η πρόβλεψη που επιστράφηκε μέσω της πρώτης κλήσης ήταν 744 για τα τρία ευρετικά και 15 για το ευρετικό κινήσεων. Το ευρετικό κινήσεων πήρε στη δεύτερη κλήση πρόβλεψη 10 και τερμάτισε την επαναληπτική διαδικασία. Το ευρετικό στόχων έκανε κλήσεις οι οποίες επέστρεψαν προβλέψεις -892,-1014,-1122,-1191,-1269,-1314,-1494,-1560 και -1601 αντίστοιχα, το ευρετικό καταστάσεων πήρε προβλέψεις -945,-1122 και -1299 και το ευρετικό συνδυασμού -837,-906,-1014,-1122,-1191 και -1371. Και στις τέσσερεις περιπτώσεις η επαναληπτική διαδικασία τελείωσε μόλις ξεπεράστηκε το κατώφλι. Ας δούμε τώρα πως αλλάζουν

τα αρχεία κινήσεων και αρχικής κατάστασης στη διαδικασία αυτή για το ευρετικό καταστάσεων. Το αρχείο κινήσεων μόλις ολοκληρωθεί η πρώτη κλήση περιέχει τα πιο κάτω δεδομένα

(move loc-x2-y2 loc-x1-y2)
(move loc-x1-y2 loc-x0-y2)
(move loc-x0-y2 loc-x0-y1)
(move loc-x0-y1 loc-x1-y1)
(move loc-x1-y1 loc-x2-y1)

Με το τέλος της τελευταίας κλήσης , με την κωδικοποίηση χωρίς ευρετικές μεθόδους περιέχει τα πιο κάτω δεδομένα.

(move loc-x2-y2 loc-x1-y2)
(move loc-x1-y2 loc-x0-y2)
(move loc-x0-y2 loc-x0-y1)
(move loc-x0-y1 loc-x1-y1)
(move loc-x1-y1 loc-x2-y1) --- Τέλος 1ης επανάληψης
(move loc-x2-y1 loc-x2-y0)
(move loc-x2-y0 loc-x3-y0)
(move loc-x3-y0 loc-x3-y1)
(move loc-x3-y1 loc-x3-y2)
(move loc-x3-y2 loc-x2-y2) --- Τέλος 2ης επανάληψης
(move loc-x2-y2 loc-x3-y2)
(move loc-x3-y2 loc-x3-y3)
(move loc-x3-y3 loc-x2-y3)
(move loc-x2-y3 loc-x1-y3)
(move loc-x1-y3 loc-x0-y3) --- Τέλος 3ης επανάληψης
(move loc-x0-y3 loc-x0-y2)
(move loc-x0-y2 loc-x0-y1)
(move loc-x0-y1 loc-x0-y0)
(move loc-x0-y0 loc-x1-y0) --- Τελευταία κλήση, χωρίς ευρετικό

Για την δημιουργία του πιο πάνω αρχείου εκτελούνταν πέντε δράσεις σε κάθε επανάληψη. Φαίνεται λοιπόν πως στη τελευταία κλήση χρειάστηκαν μόνο τέσσερεις κινήσεις για την εύρεση της λύσης. Αυτό σημαίνει πως η επαναληπτική διαδικασία τερματίστηκε αρκετά κοντά στη λύση του προβλήματος, δηλαδή η μέθοδος δούλεψε σωστά, με αποτέλεσμα να βρεθεί η λύση σε

καλύτερο χρόνο απ’ ότι θα μας έδινε η κωδικοποίηση χωρίς ευρετική μέθοδο. Ας δούμε τώρα τις διαφορές που υπάρχουν στο αρχείο της αρχικής κατάστασης. Θα ασχοληθούμε μόνο με τις παραμέτρους που αλλάζουν, δηλαδή τη θέση που βρίσκεται ο πράκτορας και τις θέσεις που επισκέφτηκε, αφού οι υπόλοιπες παράμετροι δεν αλλάζουν. Το αρχείο έχει την πιο κάτω μορφή με το τέλος της πρώτης κλήσης της επαναληπτικής διαδικασίας.

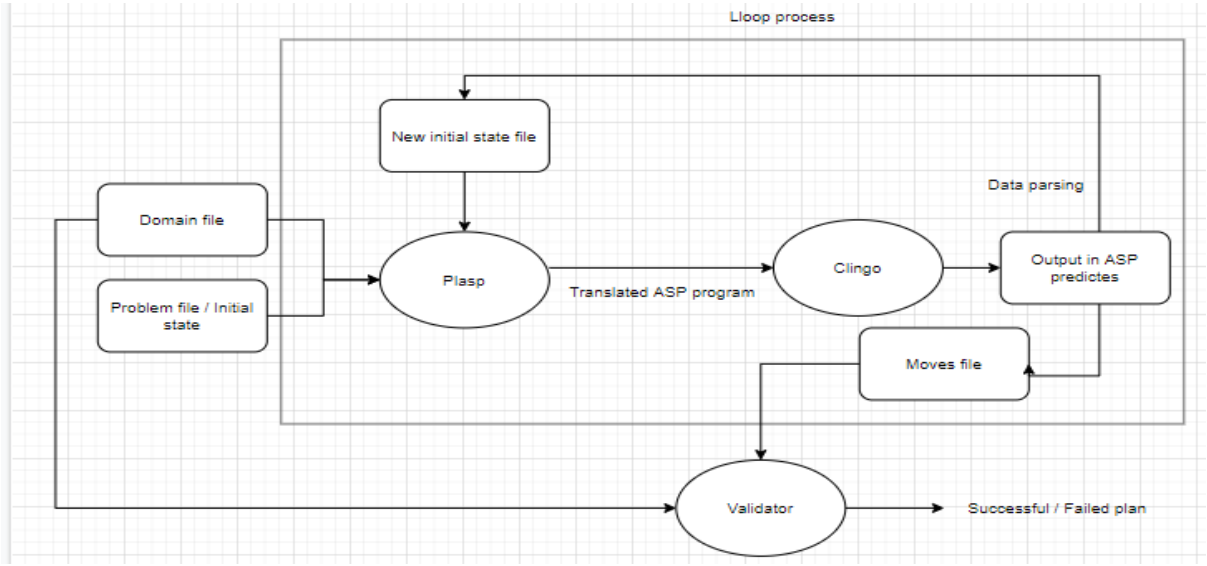
(:init
(visited loc-x2-y2)
(at-robot loc-x2-y1)
(visited loc-x2-y1)
(visited loc-x1-y2)
(visited loc-x1-y1)
(visited loc-x0-y2)
(visited loc-x0-y1))

Ο πράκτορας βρίσκεται στη τοποθεσία (2,1) και έχει επισκεφθεί 6 από τις 16 τοποθεσίες του πλέγματος. Με το τέλος της τελευταίας επαναληπτικής κλήσης , πριν την κλήση της κωδικοποίησης χωρίς ευρετικές μεθόδους περιέχει τα πιο κάτω δεδομένα.

(visited loc-x3-y2)
(visited loc-x3-y1)
(visited loc-x3-y0)
(visited loc-x2-y2)
(visited loc-x2-y1)
(visited loc-x2-y0)
(visited loc-x1-y2)
(visited loc-x1-y1)
(visited loc-x0-y2)
(visited loc-x0-y1)
(at-robot loc-x0-y3)
(visited loc-x3-y3)
(visited loc-x2-y3)
(visited loc-x1-y3)
(visited loc-x0-y3)

Ο πράκτορας έχει επισκεφθεί 14 από τις 16 τοποθεσίες. Απομένει να επισκεφθεί τις τοποθεσίες (0,0) και (1,0) έτσι ώστε να επιλύσει το πρόβλημα. Η επαναληπτική διαδικασία τερματίστηκε δηλαδή αρκετά κοντά στην τελική κατάσταση του προβλήματος. Με το πέρας της πιο πάνω διαδικασίας το domain και problem περνούν στον validator μαζί με το αρχείο κινήσεων και ο validator μας επιστρέφει αν το πλάνο που δημιουργείται μέσω τον κινήσεων είναι ένα έγκυρο πλάνο που δίνει λύση στο πρόβλημα που δόθηκε. Στην εικόνα 1 φαίνεται η διαδικασία που εκτελείται για κάθε πρόβλημα.

Εικόνα 1



4.2 Δημιουργία Ευρετικών

Αφού δημιουργήθηκε το πρόγραμμα που θα εκτελεί την επαναληπτική κλήση αποφασίσαμε να δημιουργήσουμε διάφορα ευρετικά τα οποία θα βλέπαμε κατά πόσο βελτιώνουν την απόδοση του επιλυτή σε θέμα ταχύτητας και επιλυμένων προβλημάτων. Για την πειραματική μελέτη χρησιμοποιήθηκαν 14 διαφορετικά domains και 436 προβλήματα. Κάποια domains είναι αρκετά περίπλοκα και ορισμένα προβλήματα αρκετά μεγάλου μεγέθους. Έτσι η κωδικοποίηση χωρίς ευρετικό πολλές φορές αποτυγχάνει γιατί δε μπορεί να βρει λύση στα προβλήματα αυτά. Οι ευρετικές μέθοδοι που υλοποιήσαμε είναι οι τέσσερεις μέθοδοι που αναλύθηκαν στο Κεφάλαιο 3.3.

4.3 Πειραματική Διαδικασία

Αφού υλοποιήθηκαν το πρόγραμμα και τα ευρετικά περάσαμε στη πειραματική φάση. Για τη επιτάχυνση των πειραμάτων δημιουργήθηκε αυτόματη μέθοδος εκτέλεσης των πειραμάτων. Δημιουργήθηκαν ξεχωριστά αρχεία Python για κάθε ευρετικό μέσω τον οποίων εκτελείται η επαναληπτική διαδικασία για κάθε domain και πρόβλημα .Έπειτα τα αποτελέσματα επεξεργάζονται από scripts και παράγουν αναλυτικά αποτελέσματα για τους χρόνους επίλυσης και για τον αριθμό προβλημάτων που επιλύθηκαν. Για την εκτέλεση των πειραμάτων χρησιμοποιήθηκε ο server Arcadia του τμήματος. Τα πειράματα είναι αρκετά χρονοβόρα πράγμα που μας οδήγησε στην εκτέλεση πειραμάτων με αλλαγές μόνο στον αριθμό κανονικών βημάτων. Άλλες αλλαγές που μπορούν να γίνουν είναι αύξηση ή μείωση του επιτρεπτού χρόνου εκτέλεσης του επιλυτή, δοκιμές με νέες φόρμουλες κατωφλίου για ευρετικά, αλλαγές στη φόρμουλα που δίνει αξία στον αριθμό των στόχων που έχουν επιτευχθεί (ΠΣΑ πρόγραμμα) και αύξηση ή μείωση στον αριθμό απομενόντων κινήσεων για το ευρετικό κινήσεων.

Από την πλευρά της υλοποίησης οι παράμετροι που μπορούν να αλλαχτούν στη διαδικασία αυτή είναι ο αριθμός των κανονικών βημάτων και ο μέγιστος χρόνος που μπορεί να τρέχει ο επιλυτής . Όσο αφορά τα ευρετικά μπορούν να γίνουν αλλαγές στις φόρμουλες των ορίων και στις φόρμουλες που ελαχιστοποιούν τον αριθμό των απομένων κινήσεων. Κάποια προβλήματα είναι αρκετά μεγάλα και πολύπλοκα άρα υπάρχει η πιθανότητα ο επιλυτής να πέσει σε infinite loop. Θέτοντας ένα όριο στο χρόνο που μπορεί να τρέχει ο επιλυτής λύνεται αυτό το πρόβλημα. Όταν γίνει υπέρβαση του ορίου ο επιλυτής σταματάει την εξερεύνηση και γίνεται ελέγχος των μέχρι στιγμής αποτελεσμάτων. Αν ο επιλυτής φτάσει στο μέγιστο όριο χρόνου χωρίς να βρεθεί κάποιο πλάνο τότε τερματίζει τη διαδικασία για το συγκεκριμένο πρόβλημα και επιστρέφει μήνυμα πως το πλάνο δεν μπορεί να λυθεί στο χρόνο που δόθηκε, αλλιώς επαναλαμβάνει μέχρι να βρεθεί η λύση του προβλήματος .

Κεφάλαιο 5

Πειραματική Ανάλυση

5.1 Συγκρίσεις με κωδικοποίηση χωρίς ευρετικό

5.1.1 Σύγκριση με βάση τα λυμένα προβλήματα

Αρχικά θα αναλύσουμε κατά πόσο οι ευρετικές μέθοδοι που χρησιμοποιήθηκαν αποδίδουν καλύτερα από την κωδικοποίηση χωρίς ευρετικό. Στον Πίνακα 1 φαίνονται πόσα προβλήματα λύθηκαν με το κάθε ευρετικό σε κάθε domain καθώς και ο συνολικός αριθμός των προβλημάτων που επιλύθηκαν στη κάθε κατηγορία. Στη πρώτη στήλη καταγράφεται το domain και στις υπόλοιπες πέντε στήλες το ευρετικό που χρησιμοποιήθηκε με την αντίστοιχη σειρά, κωδικοποίηση χωρίς ευρετικό, ευρετικό κινήσεων, ευρετικό συνδυασμού, ευρετικό στόχων και ευρετικό καταστάσεων. Για τα δεδομένα αυτής της φάσης χρησιμοποιήθηκαν τέσσερα κανονικά βήματα στα ευρετικά και μέγιστο timeout ανά run 600 δευτερόλεπτα ενώ για την κωδικοποίηση χωρίς ευρετικό συνολικό timeout 3600 δευτερόλεπτα.

Με βάση τον Πίνακα 1 όλα τα ευρετικά κατάφεραν να επιλύσουν περισσότερα προβλήματα από την κωδικοποίηση χωρίς ευρετικό. Σε ελάχιστα domain φαίνεται πως η κωδικοποίηση χωρίς ευρετικό έδωσε περισσότερες λύσεις από τα ευρετικά. Αυτό οφείλεται στο ότι το timeout για την κωδικοποίηση χωρίς ευρετικό έχει τεθεί σε 3600 δευτερόλεπτα, αφού γίνεται μόνο ένα ενιαίο run, ενώ στα ευρετικά λόγω της επαναληπτικής κλήσεις που γίνεται το μέγιστο timeout που μπορεί να πάρει ένα run είναι 600 δευτερόλεπτα. Έτσι κάποιες φορές μπορεί να φτάσουμε σε timeout στις ευρετικές μεθόδους πριν βρεθεί η λύση ενώ στην κωδικοποίηση χωρίς ευρετικό να βρεθεί η λύση. Σε θέμα ποσοστών η κωδικοποίηση χωρίς ευρετικό έλυσε 47.47% των προβλημάτων, το ευρετικό

κινήσεων 61.69% των προβλημάτων, το ευρετικό στόχων 63.76% των προβλημάτων, το ευρετικό καταστάσεων 59.40% των προβλημάτων και το ευρετικό συνδυασμού 61.47% των προβλημάτων.

Πίνακας 1/ Επιλύσεις προβλημάτων ανά domain

Domain	No Heuristic	Actions	Combination	Goals	States
barman	4	0	2	4	2
Depots	15	20	21	21	21
driverlog	14	18	18	18	18
elevators	10	16	19	19	19
satellite	15	20	20	21	20
transport	11	13	22	22	22
visit	11	9	14	19	13
trucks	9	5	3	3	6
storage	16	26	27	27	21
pathways	15	27	30	30	29
pipes	21	25	22	22	21
rovers	33	36	39	40	36
TPP	28	28	30	30	30
openstacks	5	26	1	2	1
Total	207	269	268	278	259

5.1.2 Σύγκριση με βάση τη ταχύτητα

Ας ελέγξουμε τώρα τις αποδόσεις που είχαμε σε θέμα ταχύτητας. Στο Πίνακα 2 βρίσκονται αναλυτικά δεδομένα όσο αφορά την ταχύτητα επίλυσης κάθε ευρετικού. Στη πρώτη στήλη φαίνεται το domain και ο αριθμός των προβλημάτων που έχει το συγκεκριμένο domain. Στη δεύτερη στήλη αναφέρονται σημεία χρόνου με τα οποία κατηγοριοποιήσαμε τις λύσεις που βρήκαμε. Συγκεκριμένα οι κατηγορίες είναι λύση μέχρι 900 δευτερόλεπτα, λύση μέχρι 1800 δευτερόλεπτα, λύση μέχρι 2700 δευτερόλεπτα, λύση μέχρι 3600 δευτερόλεπτα και λύση με περισσότερο από 3600 δευτερόλεπτα. Οι υπόλοιπες 5 στήλες καθορίζουν το ευρετικό με την αντίστοιχη σειρά, κωδικοποίηση χωρίς ευρετικό, ευρετικό κινήσεων, ευρετικό συνδυασμού, ευρετικό στόχων και ευρετικό καταστάσεων. Η γραμμή total δείχνει το συνολικό αριθμό προβλημάτων που έχουν επιλυθεί στην κάθε κατηγορία.

Βάσει των αποτελεσμάτων του Πίνακα 2 φαίνεται πως για τα πλείστα domain οι ευρετικοί μέθοδοι, εκτός του ευρετικού κινήσεων, έδωσαν αρκετά ικανοποιητικά αποτελέσματα αφού επιλύουν ίσα ή και περισσότερα προβλήματα από την κωδικοποίηση χωρίς ευρετικό σε χαμηλούς χρόνους (< 900 δευτερολέπτων) . Συγκεκριμένα η κωδικοποίηση χωρίς ευρετικό έλυσε 39.68% των συνολικών προβλημάτων σε χρόνο κάτω από 900 δευτερόλεπτα, το ευρετικό κινήσεων 25.46%, το ευρετικό συνδυασμού 50%, το ευρετικό στόχων 50.92% και το ευρετικό καταστάσεων 50.23%.

Πίνακας 2/ Ταχύτητες επίλυσης προβλημάτων

		No Heuristic	Actions	Combination	Goals	States
	< 900	0	0	2	4	2
barman	< 1800	0	0	0	0	0
	< 2700	1	0	0	0	0
40 problems	< 3600	2	0	0	0	0
	> 3600	1	0	0	0	0
	Total	4	0	2	4	2
	< 900	14	10	21	21	21
Depots	< 1800	0	4	0	0	0
	< 2700	0	1	0	0	0
22 problems	< 3600	0	2	0	0	0
	> 3600	1	3	0	0	0
	Total	15	20	21	21	21
	< 900	13	14	15	16	16
driverlog	< 1800	0	1	2	1	1
	< 2700	1	0	0	0	0
20 problems	< 3600	0	0	0	0	0
	> 3600	0	3	1	1	1
	Total	14	18	18	18	18
	< 900	9	4	10	11	12
elevators	< 1800	0	1	2	1	0
	< 2700	0	4	3	3	3
29 problems	< 3600	0	2	1	1	1
	> 3600	1	5	3	3	3
	Total	10	16	19	19	19
	< 900	13	7	13	13	13
satellite	< 1800	1	5	5	5	5
	< 2700	0	4	1	1	1
36 problems	< 3600	0	1	1	1	1
	> 3600	1	3	0	1	0
	Total	15	20	20	21	20
	< 900	9	6	13	13	13
transport	< 1800	0	1	3	2	2
	< 2700	0	1	1	2	2
29 problems	< 3600	1	0	1	1	1
	> 3600	1	5	4	4	4
	Total	11	13	22	22	22
	< 900	9	7	14	16	13
visit	< 1800	0	0	0	2	0
	< 2700	0	1	0	0	0
20 problems	< 3600	1	1	0	1	0
	> 3600	1	0	0	0	0
	Total	11	9	14	19	13

		No Heuristic	Actions	Combination	Goals	States
	< 900	6	4	3	3	6
trucks	< 1800	1	0	0	0	0
	< 2700	0	0	0	0	0
30 problems	< 3600	0	1	0	0	0
	> 3600	2	0	0	0	0
	Total	9	5	3	3	6
	< 900	15	17	21	20	20
storage	< 1800	0	1	1	2	1
	< 2700	0	1	1	1	0
30 problems	< 3600	0	3	0	2	0
	> 3600	1	4	4	2	0
	Total	16	26	27	27	21
	< 900	10	8	29	28	28
pathways	< 1800	1	1	0	1	1
	< 2700	2	2	1	1	0
30 problems	< 3600	1	0	0	0	0
	> 3600	1	16	0	0	0
	Total	15	27	30	30	29
	< 900	13	11	16	15	14
pipes	< 1800	3	3	3	4	2
	< 2700	1	3	0	0	2
50 problems	< 3600	0	6	1	1	0
	> 3600	4	2	2	2	3
	Total	21	25	22	22	21
	< 900	32	18	30	30	30
rovers	< 1800	0	12	4	6	4
	< 2700	1	0	2	1	2
40 problems	< 3600	0	1	0	0	0
	> 3600	0	5	3	3	0
	Total	33	36	39	40	36
	< 900	25	9	30	30	30
TPP	< 1800	3	4	0	0	0
	< 2700	0	2	0	0	0
30 problems	< 3600	0	10	0	0	0
	> 3600	0	3	0	0	0
	Total	28	28	30	30	30
	< 900	5	6	1	2	1
openstacks	< 1800	0	1	0	0	0
	< 2700	0	0	0	0	0
30 problems	< 3600	0	0	0	0	0
	> 3600	0	19	0	0	0
	Total	5	26	1	2	1

5.2 Συγκρίσεις μεταξύ ευρετικών μεθόδων

5.2.1 Σύγκριση τριών και τεσσάρων κανονικών βημάτων

Αφού πήραμε θετικές ενδείξεις όσο αφορά την αποδοτικότητα των ευρετικών αποφασίσαμε να προσπαθήσουμε να κάνουμε βελτιστοποιήσεις μέσω μικρών αλλαγών. Η πρώτη σειρά πειραμάτων έγινε με χρήση τεσσάρων κανονικών βημάτων στα ευρετικά. Αποφασίσαμε να δούμε πως θα επηρεαστούν οι χρόνοι και ο αριθμός των προβλημάτων που έχουν επιλυθεί αν ο αριθμός των κανονικών βημάτων μειωθεί στα τρία.

Στους πιο κάτω πίνακες 3 και 4 φαίνονται πληροφορίες σχετικά με το πόσα προβλήματα έχει λύσει επιτυχημένα το κάθε ευρετικό, πόσα προβλήματα τερματίστηκαν λόγω ότι έφτασαν στο μέγιστο επιτρεπτό χρόνο εκτέλεσης και πόσα προβλήματα τέλειωσαν με επιτυχία αλλά δεν έδωσαν σωστό πλάνο ως αποτέλεσμα. Ο πίνακας 3 περιέχει τα αποτελέσματα των πειραμάτων στα οποία εκτελούνται 3 κανονικά βήματα και ο πίνακας 4 περιέχει τα αποτελέσματα των πειραμάτων στα οποία εκτελούνται 4 κανονικά βήματα. Η κάθε στήλη υποδεικνύει τα αποτελέσματα για κάθε διαφορετικό ευρετικό. Αναλυτικά οι πιο κάτω έννοιες του πίνακα εννοούν τα εξής:

Runs finished without timeout: ο αριθμός των προβλημάτων του συγκεκριμένου Domain στα οποία βρέθηκε λύση χωρίς να υπάρχει υπέρβαση του επιτρεπτού χρόνου εκτέλεσης σε κάποια από τις εσωτερικές επαναλήψεις

Runs finished with timeout: ο αριθμός των προβλημάτων του συγκεκριμένου Domain που δεν βρέθηκε λύση αφού υπάρχει υπέρβαση του επιτρεπτού χρόνου εκτέλεσης σε κάποια από τις εσωτερικές επαναλήψεις

Runs finished with planning fail: ο αριθμός των προβλημάτων του συγκεκριμένου Domain που βρέθηκε λύση αλλά ο validator επέστρεψε ότι το πλάνο που έχει βρεθεί δεν ικανοποιεί τη λύση του προβλήματος.

Πίνακας 3/ Αριθμός κανονικών βημάτων = 3

	Actions	Combination	Goals	States
Domain: barman				
Runs finished without timeout:	1	0	0	1
Runs finished with timeout:	39	40	40	39
Runs finished with planning fail:	0	0	0	0
Domain: Depots				
Runs finished without timeout:	20	21	21	21
Runs finished with timeout:	2	1	1	1
Runs finished with planning fail:	0	0	0	0
Domain: driverlog				
Runs finished without timeout:	18	17	19	18
Runs finished with timeout:	2	3	1	2
Runs finished with planning fail:	0	0	0	0
Domain: elevators				
Runs finished without timeout:	13	19	19	19
Runs finished with timeout:	16	10	10	10
Runs finished with planning fail:	0	0	0	0
Domain: satellite				
Runs finished without timeout:	21	22	22	20
Runs finished with timeout:	15	14	14	16
Runs finished with planning fail:	0	0	0	0
Domain: transport				
Runs finished without timeout:	13	15	15	17
Runs finished with timeout:	16	14	14	12
Runs finished with planning fail:	0	0	0	0
Domain: visit				
Runs finished without timeout:	8	15	18	14
Runs finished with timeout:	12	5	2	6
Runs finished with planning fail:	0	0	0	0
Domain: trucks				
Runs finished without timeout:	4	5	5	5
Runs finished with timeout:	26	25	25	25
Runs finished with planning fail:	0	0	0	0
Domain: storage				
Runs finished without timeout:	27	27	27	21
Runs finished with timeout:	3	3	3	9
Runs finished with planning fail:	0	0	0	0
Domain: pathways				
Runs finished without timeout:	27	29	29	27
Runs finished with timeout:	3	1	1	3
Runs finished with planning fail:	0	0	0	0
Domain: pipes				
Runs finished without timeout:	20	18	19	20
Runs finished with timeout:	30	32	31	30
Runs finished with planning fail:	0	0	0	0

	Actions	Combination	Goals	States
Domain: rovers				
Runs finished without timeout:	35	39	40	37
Runs finished with timeout:	5	1	0	3
Runs finished with planning fail:	0	0	0	0
Domain: TPP				
Runs finished without timeout:	28	28	30	29
Runs finished with timeout:	2	2	0	1
Runs finished with planning fail:	0	0	0	0
Domain: openstacks				
Runs finished without timeout:	18	0	2	1
Runs finished with timeout:	7	30	28	29
Runs finished with planning fail:	5	0	0	0

Πίνακας 4/ Αριθμός κανονικών βημάτων = 4

	Actions	Combination	Goals	States
Domain: barman				
Runs finished without timeout:	0	2	4	2
Runs finished with timeout:	40	38	36	38
Runs finished with planning fail:	0	0	0	0
Domain: Depots				
Runs finished without timeout:	20	21	21	21
Runs finished with timeout:	2	1	1	1
Runs finished with planning fail:	0	0	0	0
Domain: driverlog				
Runs finished without timeout:	18	18	18	18
Runs finished with timeout:	2	2	2	2
Runs finished with planning fail:	0	0	0	0
Domain: elevators				
Runs finished without timeout:	16	19	19	19
Runs finished with timeout:	13	10	10	10
Runs finished with planning fail:	0	0	0	0
Domain: satellite				
Runs finished without timeout:	20	20	21	20
Runs finished with timeout:	16	16	15	16
Runs finished with planning fail:	0	0	0	0
Domain: transport				
Runs finished without timeout:	13	22	22	22
Runs finished with timeout:	16	7	7	7
Runs finished with planning fail:	0	0	0	0
Domain: visit				
Runs finished without timeout:	9	14	19	13
Runs finished with timeout:	11	6	1	7
Runs finished with planning fail:	0	0	0	0

	Actions	Combination	Goals	States
Domain: trucks				
Runs finished without timeout:	5	3	3	6
Runs finished with timeout:	25	27	27	24
Runs finished with planning fail:	0	0	0	0
Domain: storage				
Runs finished without timeout:	26	27	27	21
Runs finished with timeout:	4	3	3	9
Runs finished with planning fail:	0	0	0	0
Domain: pathways				
Runs finished without timeout:	27	30	30	29
Runs finished with timeout:	3	0	0	1
Runs finished with planning fail:	0	0	0	0
Domain: pipes				
Runs finished without timeout:	25	22	22	21
Runs finished with timeout:	25	28	28	29
Runs finished with planning fail:	0	0	0	0
Domain: rovers				
Runs finished without timeout:	36	39	40	36
Runs finished with timeout:	4	1	0	4
Runs finished with planning fail:	0	0	0	0
Domain: TPP				
Runs finished without timeout:	28	30	30	30
Runs finished with timeout:	2	0	0	0
Runs finished with planning fail:	0	0	0	0
Domain: openstacks				
Runs finished without timeout:	26	1	2	1
Runs finished with timeout:	4	29	28	29
Runs finished with planning fail:	0	0	0	0

Βάσει των πιο πάνω δεδομένων φαίνεται πως η συμπεριφορά των ευρετικών είναι παρόμοια σχεδόν σε όλα τα domain. Επιλύουν κοντινό αριθμό προβλημάτων χωρίς timeout με μόνη εξαίρεση το domain openstacks στο οποίο το ευρετικό κινήσεων είναι το μόνο που καταφέρνει να επιλύσει τα περισσότερα προβλήματα χωρίς να φτάσει σε timeout. Αυτό οφείλεται στο τρόπο με τον οποίο λειτουργεί το ευρετικό αφού ψάχνει κάθε φορά τρόπο να ελαχιστοποιήσει τις απομένοντες κινήσεις για επίλυση του προβλήματος. Αυτή η λογική είναι αρκετά χρονοβόρα με αποτέλεσμα κάποιες φορές να οδηγεί σε τεράστιους χρόνους εκτέλεσης σε σύγκριση με τα άλλα ευρετικά. Φαίνεται όμως πως αποτελεί μια πιο ασφαλή μέθοδο για πολύπλοκα domain.

Στο πίνακα 5 φαίνονται τα αποτελέσματα των πινάκων 3 και 4 συγκεντρωμένα έτσι ώστε να γίνει σύγκριση της επιρροής των κανονικών βημάτων στο αποτέλεσμα. Σε κάθε γραμμή φαίνεται ο

αριθμός των επιτυχημένων προβλημάτων που έλυσε το κάθε ευρετικό στο κάθε domain. Στο πίνακα καταγράφονται διαδοχικά πρώτα οι τιμές με 3 κανονικά βήματα και μετά με 4 κανονικά βήματα για κάθε ευρετικό.

Πίνακας 5/ Συγκριτικά αποτελέσματα.

	Actions/3	Actions/4	Combination/3	Combination/4	Goals/3	Goals/4	States/3	States/4
barman	1	0	0	2	0	4	1	2
Depots	20	20	21	21	21	21	21	21
driverlog	18	18	17	18	19	18	18	18
elevators	13	16	19	19	19	19	19	19
satellite	21	20	22	20	22	21	20	20
transport	13	13	15	22	15	22	17	22
visit	8	9	15	14	18	19	14	13
trucks	4	5	5	3	5	3	5	6
storage	27	26	27	27	27	27	21	21
pathways	27	27	29	30	29	30	27	29
pipes	20	25	18	22	19	22	20	21
rovers	35	36	39	39	40	40	37	36
TPP	28	28	28	30	30	30	29	30
openstacks	18	26	0	1	2	2	1	1
Total	253	269	255	268	266	278	250	259

Συγκεντρωτικά το ευρετικό κινήσεων με τρία κανονικά βήματα έλυσε 58.03% των προβλημάτων ενώ με τέσσερα κανονικά βήματα 61.70%, το ευρετικό συνδυασμού με τρία κανονικά βήματα έλυσε 58.49% των προβλημάτων ενώ με τέσσερα κανονικά βήματα 61.47%,το ευρετικό στόχων με τρία κανονικά βήματα έλυσε 61.00% των προβλημάτων ενώ με τέσσερα κανονικά βήματα 63.76% και το ευρετικό καταστάσεων με τρία κανονικά βήματα έλυσε 57.34% των προβλημάτων ενώ με τέσσερα κανονικά βήματα 59.40%. Με βάση τα πιο πάνω αποτελέσματα βλέπουμε πως όταν εκτελούνται τέσσερα κανονικά βήματα έχουμε περισσότερες επιτυχημένες επιλύσεις των προβλημάτων παρά όταν εκτελούνται τρία κανονικά βήματα σε όλα τα ευρετικά. Το ευρετικό στόχων φαίνεται να αποδίδει καλύτερα από τα υπόλοιπα ευρετικά και στις δύο περιπτώσεις . Η διαφορά του αριθμού των επιπλέον προβλημάτων που επιλύονται ανά domain όταν χρησιμοποιηθούν τέσσερα κανονικά βήματα είναι περίπου 1-5 προβλήματα στις πλείστες περιπτώσεις. Η μόνη περίπτωση που φαίνεται να υπάρχει μεγάλη διαφορά είναι στο domain openstacks με τη χρήση του ευρετικού κινήσεων όπου επιλύονται 8 περισσότερα προβλήματα. Αυτό μπορεί να οφείλεται στο τρόπο με τον οποίο δουλεύει το ευρετικό αφού μια περισσότερη κίνηση μπορεί να οδηγήσει σε τεράστιες διαφορές στο πως θα εξερευνηθούν αργότερα οι πιθανές επόμενες χαλαρωμένες κινήσεις. Το ευρετικό καταστάσεων φαίνεται πως δεν είχε μεγάλη αύξηση στο ποσοστό επίλυσης, είχε μόλις 9 περισσότερα επιλυμένα προβλήματα ενώ τα υπόλοιπα ευρετικά περισσότερα από 12, σύγκριση με τα υπόλοιπα ευρετικά το οποίο είναι ενδεικτικό στοιχείο στο ότι χρειάζεται βελτίωση η φόρμουλα στο κατώφλι.

Στου πίνακες 6 και 7 καταγράφονται πόσα προβλήματα λύθηκαν ανά domain σε 5 διαφορετικά χρονικά πλαίσια, λιγότερο από 900,1800,2700,3600 δευτερόλεπτα και περισσότερο από 3600 δευτερόλεπτα. Στη πρώτη στήλη αναγράφεται το domain και ο αριθμός των προβλημάτων που έχει το domain, στη δεύτερη στήλη αναγράφονται οι κατηγορίες χρόνων επίλυσης που ορίσαμε και ο συνολικός αριθμός επιλυμένων προβλημάτων που επιλύθηκαν στο συγκεκριμένο domain. Οι υπόλοιπες στήλες αναφέρονται στις επιλύσεις που επιτεύχθηκαν από κάθε ευρετικό στην κάθε κατηγορία .

Πίνακας 6 / Ταχύτητες επίλυσης c1=3

		Actions	Combination	Goals	States
	< 900	0	0	0	1
barman	< 1800	0	0	0	0
	< 2700	1	0	0	0
40 problems	< 3600	0	0	0	0
	> 3600	0	0	0	0
	Total	1	0	0	1
	< 900	10	21	21	21
Depots	< 1800	4	0	0	0
	< 2700	0	0	0	0
22 problems	< 3600	1	0	0	0
	> 3600	5	0	0	0
	Total	20	21	21	21
	< 900	15	17	17	17
driverlog	< 1800	0	0	2	1
	< 2700	0	0	0	0
20 problems	< 3600	0	0	0	0
	> 3600	3	0	0	0
	Total	18	17	19	18
	< 900	3	10	10	10
elevators	< 1800	2	2	2	2
	< 2700	2	0	0	0
29 problems	< 3600	1	3	1	3
	> 3600	5	4	6	4
	Total	13	19	19	19
	< 900	7	15	17	15
satellite	< 1800	4	3	1	4
	< 2700	2	2	2	1
36 problems	< 3600	1	0	0	0
	> 3600	7	2	2	0
	Total	21	22	22	20
	< 900	6	10	8	11
transport	< 1800	2	0	0	1
	< 2700	0	2	1	2
29 problems	< 3600	0	0	1	0
	> 3600	5	3	5	3
	Total	13	15	15	17

		Actions	Combination	Goals	States
	< 900	7	15	18	14
visit	< 1800	0	0	0	0
	< 2700	0	0	0	0
20 problems	< 3600	0	0	0	0
	> 3600	1	0	0	0
	Total	8	15	18	14
	< 900	4	5	5	5
trucks	< 1800	0	0	0	0
	< 2700	0	0	0	0
30 problems	< 3600	0	0	0	0
	> 3600	0	0	0	0
	Total	4	5	5	5
	< 900	18	22	22	20
storage	< 1800	0	2	1	1
	< 2700	0	0	1	0
30 problems	< 3600	2	1	0	0
	> 3600	7	2	3	0
	Total	27	27	27	21
	< 900	6	27	28	26
pathways	< 1800	3	1	1	0
	< 2700	1	1	0	1
30 problems	< 3600	0	0	0	0
	> 3600	17	0	0	0
	Total	27	29	29	27
	< 900	11	13	13	13
pipes	< 1800	2	2	3	3
	< 2700	0	1	1	1
50 problems	< 3600	3	0	0	0
	> 3600	4	2	2	3
	Total	20	18	19	20
	< 900	15	31	31	31
rovers	< 1800	10	3	3	2
	< 2700	3	2	3	2
40 problems	< 3600	0	0	1	1
	> 3600	7	3	2	1
	Total	35	39	40	37

		Actions	Combination	Goals	States
	< 900	8	27	30	29
TPP	< 1800	7	1	0	0
	< 2700	0	0	0	0
30 problems	< 3600	3	0	0	0
	> 3600	10	0	0	0
	Total	28	28	30	29
	< 900	5	0	2	1
openstacks	< 1800	0	0	0	0
	< 2700	0	0	0	0
30 problems	< 3600	0	0	0	0
	> 3600	13	0	0	0
	Total	18	0	2	1

Πίνακας 7 / Ταχύτητες επίλυσης c1=4

		Actions	Combination	Goals	States
	< 900	0	2	4	2
barman	< 1800	0	0	0	0
	< 2700	0	0	0	0
40 problems	< 3600	0	0	0	0
	> 3600	0	0	0	0
	Total	0	2	4	2
	< 900	10	21	21	21
Depots	< 1800	4	0	0	0
	< 2700	1	0	0	0
22 problems	< 3600	2	0	0	0
	> 3600	3	0	0	0
	Total	20	21	21	21
	< 900	14	15	16	16
driverlog	< 1800	1	2	1	1
	< 2700	0	0	0	0
20 problems	< 3600	0	0	0	0
	> 3600	3	1	1	1
	Total	18	18	18	18
	< 900	4	10	11	12
elevators	< 1800	1	2	1	0
	< 2700	4	3	3	3
29 problems	< 3600	2	1	1	1
	> 3600	5	3	3	3
	Total	16	19	19	19

		Actions	Combination	Goals	States
	< 900	7	13	13	13
satellite	< 1800	5	5	5	5
	< 2700	4	1	1	1
36 problems	< 3600	1	1	1	1
	> 3600	3	0	1	0
	Total	20	20	21	20
	< 900	6	13	13	13
transport	< 1800	1	3	2	2
	< 2700	1	1	2	2
29 problems	< 3600	0	1	1	1
	> 3600	5	4	4	4
	Total	13	22	22	22
	< 900	7	14	16	13
visit	< 1800	0	0	2	0
	< 2700	1	0	0	0
20 problems	< 3600	1	0	1	0
	> 3600	0	0	0	0
	Total	9	14	19	13
	< 900	4	3	3	6
trucks	< 1800	0	0	0	0
	< 2700	0	0	0	0
30 problems	< 3600	1	0	0	0
	> 3600	0	0	0	0
	Total	5	3	3	6
	< 900	17	21	20	20
storage	< 1800	1	1	2	1
	< 2700	1	1	1	0
30 problems	< 3600	3	0	2	0
	> 3600	4	4	2	0
	Total	26	27	27	21
	< 900	8	29	28	28
pathways	< 1800	1	0	1	1
	< 2700	2	1	1	0
30 problems	< 3600	0	0	0	0
	> 3600	16	0	0	0
	Total	27	30	30	29

		Actions	Combination	Goals	States
	< 900	11	16	15	14
pipes	< 1800	3	3	4	2
	< 2700	3	0	0	2
50 problems	< 3600	6	1	1	0
	> 3600	2	2	2	3
	Total	25	22	22	21
	< 900	18	30	30	30
rovers	< 1800	12	4	6	4
	< 2700	0	2	1	2
40 problems	< 3600	1	0	0	0
	> 3600	5	3	3	0
	Total	36	39	40	36
	< 900	9	30	30	30
TPP	< 1800	4	0	0	0
	< 2700	2	0	0	0
30 problems	< 3600	10	0	0	0
	> 3600	3	0	0	0
	Total	28	30	30	30
	< 900	6	1	2	1
openstacks	< 1800	1	0	0	0
	< 2700	0	0	0	0
30 problems	< 3600	0	0	0	0
	> 3600	19	0	0	0
	Total	26	1	2	1

Για την σύγκριση των αποτελεσμάτων θα ασχοληθούμε με τους χρόνους επίλυσης κάτω από 900 δευτερόλεπτα αφού για τα πλείστα domain αποτελούν το μεγαλύτερο ποσοστό των προβλημάτων που επιλύθηκαν. Το ευρετικό κινήσεων έλυσε 26.38% των προβλημάτων σε χρόνο κάτω από 900 δευτερόλεπτα με τρία κανονικά βήματα και 27.75% των προβλημάτων με τέσσερα κανονικά βήματα. Το ευρετικό στόχων έλυσε 50.92% των προβλημάτων και στις δύο περιπτώσεις. Το ευρετικό καταστάσεων έλυσε 49.08% των προβλημάτων με τρία κανονικά βήματα και 50.23% με τέσσερα κανονικά βήματα. Το ευρετικό συνδυασμού έλυσε 48.85% των προβλημάτων με τρία κανονικά βήματα και 50% με τέσσερα κανονικά βήματα. Φαίνεται λοιπόν πως ο χρόνος επίλυσης προβλημάτων δεν επηρεάστηκε ιδιαίτερα με την αύξηση ενός βήματος.

5.2.2 Σύγκριση τεσσάρων και πέντε κανονικών βημάτων

Με βάση τα αποτελέσματα που πήραμε μέσω της σύγκρισης τριών και τεσσάρων κανονικών βημάτων αποφασίσαμε να δούμε πως θα επηρεαστούν οι χρόνοι και ο αριθμός των προβλημάτων που έχουν επιλυθεί αν ο αριθμός των κανονικών βημάτων αυξηθεί στα πέντε.

Στους πιο κάτω πίνακες 8 και 9 φαίνονται πληροφορίες σχετικά με το πόσα προβλήματα έχει λύσει επιτυχημένα το κάθε ευρετικό, πόσα προβλήματα τερματίστηκαν λόγω ότι έφτασαν στο μέγιστο επιτρεπτό χρόνο εκτέλεσης και πόσα προβλήματα τέλειωσαν με επιτυχία αλλά δεν έδωσαν σωστό πλάνο ως αποτέλεσμα. Ο πίνακας 8 περιέχει τα αποτελέσματα των πειραμάτων στα οποία εκτελούνται 4 κανονικά βήματα και ο πίνακας 9 περιέχει τα αποτελέσματα των πειραμάτων στα οποία εκτελούνται 5 κανονικά βήματα.

Πίνακας 8/ Αριθμός κανονικών βημάτων = 4

	Actions	Combination	Goals	States
Domain: barman				
Runs finished without timeout:	0	2	4	2
Runs finished with timeout:	40	38	36	38
Runs finished with planning fail:	0	0	0	0
Domain: Depots				
Runs finished without timeout:	20	21	21	21
Runs finished with timeout:	2	1	1	1
Runs finished with planning fail:	0	0	0	0
Domain: driverlog				
Runs finished without timeout:	18	18	18	18
Runs finished with timeout:	2	2	2	2
Runs finished with planning fail:	0	0	0	0
Domain: elevators				
Runs finished without timeout:	16	19	19	19
Runs finished with timeout:	13	10	10	10
Runs finished with planning fail:	0	0	0	0
Domain: satellite				
Runs finished without timeout:	20	20	21	20
Runs finished with timeout:	16	16	15	16
Runs finished with planning fail:	0	0	0	0
Domain: transport				
Runs finished without timeout:	13	22	22	22
Runs finished with timeout:	16	7	7	7
Runs finished with planning fail:	0	0	0	0
Domain: visit				
Runs finished without timeout:	9	14	19	13
Runs finished with timeout:	11	6	1	7
Runs finished with planning fail:	0	0	0	0

	Actions	Combination	Goals	States
Domain: trucks				
Runs finished without timeout:	5	3	3	6
Runs finished with timeout:	25	27	27	24
Runs finished with planning fail:	0	0	0	0
Domain: storage				
Runs finished without timeout:	26	27	27	21
Runs finished with timeout:	4	3	3	9
Runs finished with planning fail:	0	0	0	0
Domain: pathways				
Runs finished without timeout:	27	30	30	29
Runs finished with timeout:	3	0	0	1
Runs finished with planning fail:	0	0	0	0
Domain: pipes				
Runs finished without timeout:	25	22	22	21
Runs finished with timeout:	25	28	28	29
Runs finished with planning fail:	0	0	0	0
Domain: rovers				
Runs finished without timeout:	36	39	40	36
Runs finished with timeout:	4	1	0	4
Runs finished with planning fail:	0	0	0	0
Domain: TPP				
Runs finished without timeout:	28	30	30	30
Runs finished with timeout:	2	0	0	0
Runs finished with planning fail:	0	0	0	0
Domain: openstacks				
Runs finished without timeout:	26	1	2	1
Runs finished with timeout:	4	29	28	29
Runs finished with planning fail:	0	0	0	0

Πίνακας 9/ Αριθμός κανονικών βημάτων =5

	Actions	Combination	Goals	States
Domain: barman				
Runs finished without timeout:	1	5	6	4
Runs finished with timeout:	39	35	34	36
Runs finished with planning fail:	0	0	0	0
Domain: Depots				
Runs finished without timeout:	21	21	21	21
Runs finished with timeout:	1	1	1	1
Runs finished with planning fail:	0	0	0	0
Domain: driverlog				
Runs finished without timeout:	17	17	17	17
Runs finished with timeout:	3	3	3	3
Runs finished with planning fail:	0	0	0	0

Domain: elevators				
Runs finished without timeout:	18	19	19	19
Runs finished with timeout:	11	10	10	10
Runs finished with planning fail:	0	0	0	0
Domain: satellite				
Runs finished without timeout:	19	19	19	19
Runs finished with timeout:	17	17	17	17
Runs finished with planning fail:	0	0	0	0
Domain: transport				
Runs finished without timeout:	17	22	21	21
Runs finished with timeout:	12	7	8	8
Runs finished with planning fail:	0	0	0	0
Domain: visit				
Runs finished without timeout:	10	14	15	15
Runs finished with timeout:	10	6	5	5
Runs finished with planning fail:	0	0	0	0
Domain: trucks				
Runs finished without timeout:	7	8	7	9
Runs finished with timeout:	23	22	23	21
Runs finished with planning fail:	0	0	0	0
Domain: storage				
Runs finished without timeout:	22	22	22	21
Runs finished with timeout:	8	8	8	9
Runs finished with planning fail:	0	0	0	0
Domain: pathways				
Runs finished without timeout:	29	30	30	29
Runs finished with timeout:	1	0	0	1
Runs finished with planning fail:	0	0	0	0
Domain: pipes				
Runs finished without timeout:	23	23	24	22
Runs finished with timeout:	27	27	26	28
Runs finished with planning fail:	0	0	0	0
Domain: rovers				
Runs finished without timeout:	39	39	39	38
Runs finished with timeout:	1	1	1	2
Runs finished with planning fail:	0	0	0	0
Domain: TPP				
Runs finished without timeout:	30	30	30	30
Runs finished with timeout:	0	0	0	0
Runs finished with planning fail:	0	0	0	0
Domain: openstacks				
Runs finished without timeout:	26	3	6	4
Runs finished with timeout:	4	27	24	26
Runs finished with planning fail:	0	0	0	0

Βάσει των πιο πάνω δεδομένων φαίνεται πως η συμπεριφορά των ευρετικών είναι και πάλι παρόμοια σχεδόν σε όλα τα domain. Αυτή τη φορά το ευρετικό κινήσεων έχει φτάσει ακόμη πιο κοντά στα αποτελέσματα των υπόλοιπων ευρετικών και εξακολουθεί να δίνει καλύτερη λύση για το domain openstacks.

Στο πίνακα 10 φαίνονται τα αποτελέσματα των πιο πινάκων 8 και 9 συγκεντρωμένα έτσι ώστε να γίνει σύγκριση της επιρροής των κανονικών βημάτων στο αποτέλεσμα. Σε κάθε γραμμή φαίνεται ο αριθμός των επιτυχημένων προβλημάτων που έλυσε το κάθε ευρετικό στο κάθε domain. Στο πίνακα καταγράφονται διαδοχικά πρώτα οι τιμές με 3 κανονικά βήματα και μετά με 4 κανονικά βήματα για κάθε ευρετικό.

Πίνακας 10/ Συγκριτικά Αποτελέσματα

	Actions/4	Actions/5	Combination/4	Combination/5	Goals/4	Goals/5	States/4	States/5
barman	0	1	2	5	4	6	2	4
Depots	20	21	21	21	21	21	21	21
driverlog	18	17	18	17	18	17	18	17
elevators	16	18	19	19	19	19	19	19
satellite	20	19	20	19	21	19	20	19
transport	13	17	22	22	22	21	22	21
visit	9	10	14	14	19	15	13	15
trucks	5	7	3	8	3	7	6	9
storage	26	22	27	22	27	22	21	21
pathways	27	29	30	30	30	30	29	29
pipes	25	23	22	23	22	24	21	22
rovers	36	39	39	39	40	39	36	38
TPP	28	30	30	30	30	30	30	30
openstacks	26	26	1	3	2	6	1	4
Total	269	279	268	272	278	276	259	269

Συγκεντρωτικά το ευρετικό κινήσεων με πέντε κανονικά βήματα έλυσε 63.99% των προβλημάτων ενώ με τέσσερα κανονικά βήματα 61.70%, το ευρετικό συνδυασμού με πέντε κανονικά βήματα έλυσε 62.38% των προβλημάτων ενώ με τέσσερα κανονικά βήματα 61.47%,το ευρετικό στόχων με πέντε κανονικά βήματα έλυσε 63.30% των προβλημάτων ενώ με τέσσερα κανονικά βήματα 63.76% και το ευρετικό καταστάσεων με πέντε κανονικά βήματα έλυσε 61.69% των προβλημάτων ενώ με τέσσερα κανονικά βήματα 59.40%. Γενικά φαίνεται πως υπάρχει βελτίωση με την εκτέλεση πέντε κανονικών βημάτων εχτός στο ευρετικό στόχων. Αυτό βέβαια μπορεί να οφείλεται και στη μηχανή που έτρεξαν τα πειράματα αφού είναι τόσο μικρή η διαφορά που πιθανόν να υπάρχει λόγο υπερφόρτωσης της μηχανής κατά την επίλυση κάποιων προβλημάτων

Στου πίνακες 11 και 12 καταγράφονται πόσα προβλήματα λύθηκαν ανά domain σε 5 διαφορετικά χρονικά πλαίσια, λιγότερο από 900,1800,2700,3600 δευτερόλεπτα και περισσότερο από 3600 δευτερόλεπτα. Στη πρώτη στήλη αναγράφεται το domain και ο αριθμός των προβλημάτων που έχει το domain, στη δεύτερη στήλη αναγράφονται οι κατηγορίες χρόνων επίλυσης που ορίσαμε και ο συνολικός αριθμός επιλυμένων προβλημάτων που επιλύθηκαν στο συγκεκριμένο domain. Οι υπόλοιπες στήλες αναφέρονται στις επιλύσεις που επιτεύχθηκαν από κάθε ευρετικό στην κάθε κατηγορία .

Πίνακας 11 / Ταχύτητες επίλυσης c1=4

		Actions	Combination	Goals	States
	< 900	0	2	4	2
barman	< 1800	0	0	0	0
	< 2700	0	0	0	0
40 problems	< 3600	0	0	0	0
	> 3600	0	0	0	0
	Total	0	2	4	2
	< 900	10	21	21	21
Depots	< 1800	4	0	0	0
	< 2700	1	0	0	0
22 problems	< 3600	2	0	0	0
	> 3600	3	0	0	0
	Total	20	21	21	21
	< 900	14	15	16	16
driverlog	< 1800	1	2	1	1
	< 2700	0	0	0	0
20 problems	< 3600	0	0	0	0
	> 3600	3	1	1	1
	Total	18	18	18	18

		Actions	Combination	Goals	States
	< 900	4	10	11	12
elevators	< 1800	1	2	1	0
	< 2700	4	3	3	3
29 problems	< 3600	2	1	1	1
	> 3600	5	3	3	3
	Total	16	19	19	19
	< 900	7	13	13	13
satellite	< 1800	5	5	5	5
	< 2700	4	1	1	1
36 problems	< 3600	1	1	1	1
	> 3600	3	0	1	0
	Total	20	20	21	20
	< 900	6	13	13	13
transport	< 1800	1	3	2	2
	< 2700	1	1	2	2
29 problems	< 3600	0	1	1	1
	> 3600	5	4	4	4
	Total	13	22	22	22
	< 900	7	14	16	13
visit	< 1800	0	0	2	0
	< 2700	1	0	0	0
20 problems	< 3600	1	0	1	0
	> 3600	0	0	0	0
	Total	9	14	19	13
	< 900	4	3	3	6
trucks	< 1800	0	0	0	0
	< 2700	0	0	0	0
30 problems	< 3600	1	0	0	0
	> 3600	0	0	0	0
	Total	5	3	3	6
	< 900	17	21	20	20
storage	< 1800	1	1	2	1
	< 2700	1	1	1	0
30 problems	< 3600	3	0	2	0
	> 3600	4	4	2	0
	Total	26	27	27	21
	< 900	8	29	28	28
pathways	< 1800	1	0	1	1
	< 2700	2	1	1	0
30 problems	< 3600	0	0	0	0
	> 3600	16	0	0	0
	Total	27	30	30	29

		Actions	Combination	Goals	States
	< 900	11	16	15	14
pipes	< 1800	3	3	4	2
	< 2700	3	0	0	2
50 problems	< 3600	6	1	1	0
	> 3600	2	2	2	3
	Total	25	22	22	21
	< 900	18	30	30	30
rovers	< 1800	12	4	6	4
	< 2700	0	2	1	2
40 problems	< 3600	1	0	0	0
	> 3600	5	3	3	0
	Total	36	39	40	36
	< 900	9	30	30	30
TPP	< 1800	4	0	0	0
	< 2700	2	0	0	0
30 problems	< 3600	10	0	0	0
	> 3600	3	0	0	0
	Total	28	30	30	30
	< 900	6	1	2	1
openstacks	< 1800	1	0	0	0
	< 2700	0	0	0	0
30 problems	< 3600	0	0	0	0
	> 3600	19	0	0	0
	Total	26	1	2	1

Πίνακας 12 / Ταχύτητες επίλυσης c1=5

		Actions	Combination	Goals	States
	< 900	1	4	6	4
barman	< 1800	0	1	0	0
	< 2700	0	0	0	0
40 problems	< 3600	0	0	0	0
	> 3600	0	0	0	0
	Total	1	5	6	4
	< 900	19	19	19	19
Depots	< 1800	1	1	2	1
	< 2700	1	1	0	1
22 problems	< 3600	0	0	0	0
	> 3600	0	0	0	0
	Total	21	21	21	21
	< 900	15	15	15	15
driverlog	< 1800	1	1	1	1
	< 2700	1	1	1	1
20 problems	< 3600	0	0	0	0
	> 3600	0	0	0	0
	Total	17	17	17	17

		Actions	Combination	Goals	States
	< 900	12	12	11	12
elevators	< 1800	1	1	1	1
	< 2700	3	3	3	3
29 problems	< 3600	0	0	1	0
	> 3600	2	3	3	3
	Total	18	19	19	19
	< 900	12	12	12	12
satellite	< 1800	3	3	4	3
	< 2700	2	2	1	2
36 problems	< 3600	1	1	2	1
	> 3600	1	1	0	1
	Total	19	19	19	19
	< 900	13	13	13	13
transport	< 1800	1	2	2	2
	< 2700	2	3	2	2
29 problems	< 3600	1	1	1	1
	> 3600	0	3	3	3
	Total	17	22	21	21
	< 900	10	13	15	14
visit	< 1800	0	0	0	0
	< 2700	0	0	0	0
20 problems	< 3600	0	1	0	1
	> 3600	0	0	0	0
	Total	10	14	15	15
	< 900	7	8	7	9
trucks	< 1800	0	0	0	0
	< 2700	0	0	0	0
30 problems	< 3600	0	0	0	0
	> 3600	0	0	0	0
	Total	7	8	7	9
	< 900	18	18	18	18
storage	< 1800	2	2	1	2
	< 2700	1	1	2	1
30 problems	< 3600	0	0	0	0
	> 3600	1	1	1	0
	Total	22	22	22	21
	< 900	25	25	24	25
pathways	< 1800	3	4	3	4
	< 2700	0	0	2	0
30 problems	< 3600	0	0	0	0
	> 3600	1	1	1	0
	Total	29	30	30	29

		Actions	Combination	Goals	States
	< 900	13	15	16	15
pipes	< 1800	7	5	5	4
	< 2700	1	0	0	0
50 problems	< 3600	0	0	0	0
	> 3600	2	3	3	3
	Total	23	23	24	22
	< 900	30	30	30	30
rovers	< 1800	4	4	4	4
	< 2700	2	2	2	2
40 problems	< 3600	1	0	0	0
	> 3600	2	3	3	2
	Total	39	39	39	38
	< 900	30	30	30	30
TPP	< 1800	0	0	0	0
	< 2700	0	0	0	0
30 problems	< 3600	0	0	0	0
	> 3600	0	0	0	0
	Total	30	30	30	30
	< 900	4	3	6	4
openstacks	< 1800	16	0	0	0
	< 2700	3	0	0	0
30 problems	< 3600	1	0	0	0
	> 3600	2	0	0	0
	Total	26	3	6	4

Για την σύγκριση των αποτελεσμάτων θα ασχοληθούμε και πάλι με τους χρόνους επίλυσης κάτω από 900 δευτερόλεπτα. Το ευρετικό κινήσεων έλυσε 47.94% των προβλημάτων σε χρόνο κάτω από 900 δευτερόλεπτα με πέντε κανονικά βήματα και 27.75% των προβλημάτων με τέσσερα κανονικά βήματα. Το ευρετικό στόχων έλυσε 50.92% των προβλημάτων και στις δύο περιπτώσεις. Το ευρετικό καταστάσεων έλυσε 50.22% των προβλημάτων με πέντε κανονικά βήματα και 50.23% με τέσσερα κανονικά βήματα. Το ευρετικό συνδυασμού έλυσε 49.77% των προβλημάτων με πέντε κανονικά βήματα και 50% με τέσσερα κανονικά βήματα. Φαίνεται πως τα τρία ευρετικά έχουν πολύ κοντινές τιμές με τις τιμές και στις δύο περιπτώσεις ενώ το ευρετικό κινήσεων έχει σχεδόν διπλασιάσει την αποδοτικότητα του. Αυτό οφείλεται στον τρόπο με τον οποίο λειτουργεί. Μια επιπλέον κίνηση μπορεί να μειώσει το χρόνο εξερεύνησης μέχρι τη βέλτιστη λύση δραματικά στο ευρετικό κινήσεων.

Κεφάλαιο 6

Συμπεράσματα

6.1 Συμπεράσματα Εργασίας

Λαμβάνοντας υπόψη όσα αναφέρθηκαν πιο πάνω μπορούμε να επιβεβαιώσουμε ότι η θεωρία για τα ευρετικά είναι ορθή αφού είχαμε θετικά αποτελέσματα σε όλα τα πειράματα με ευρετικά. Βάσει των αποτελεσμάτων μπορούμε να εξάγουμε το συμπέρασμα ότι το ευρετικό στόχων είναι το πιο αποδοτικό από τα τέσσερα ευρετικά που έχουν δοκιμαστεί αφού έχει τις περισσότερες επιτυχημένες λύσεις αλλά και τους καλύτερους χρόνους. Αν ο στόχος μας είναι η επιτυχημένη επίλυση προβλημάτων και όχι η γρήγορη επίλυση τους, τότε μπορεί να γίνει συνδυασμός χρήσης των ευρετικών στόχων και κινήσεων. Αρχικά θα τρέχει το ευρετικό στόχων για κάθε πρόβλημα. Αν δεν φτάσει σε τερματισμό λόγω υπέρβασης χρόνου τότε θα προχωράμε στο επόμενο πρόβλημα. Σε περίπτωση που τερματίσει λόγω υπέρβασης χρόνου θα δοκιμάζετε ξανά το ίδιο πρόβλημα με το ευρετικό κινήσεων με διπλάσιο όριο χρόνου. Αυτό μπορεί να βρει λύσεις σε domains που είναι αρκετά περίπλοκα, όπως το domain openstacks, στα οποία αποτυγχάνουν τα υπόλοιπα ευρετικά που δοκιμάσαμε. Αν μας ενδιαφέρει η ταχύτητα η χρήση του ευρετικού στόχων είναι αρκετή. Επίσης φαίνεται πως το ευρετικό κινήσεων μπορεί να επιφέρει αρκετά καλά αποτελέσματα με ελάχιστες βελτιστοποιήσεις αφού με την αύξηση των κανονικών βημάτων κατάφερε να επιλύσει τον μεγαλύτερο αριθμό προβλημάτων σε καλούς χρόνους.

6.2 Μελλοντική Εργασία

Λόγο της μεγάλης διάρκειας χρόνου που χρειάζονται να εκτελεστούν τα πειράματα καθώς και το χρόνο που χρειάστηκε να δημιουργηθεί η όλη διαδικασία δεν έγιναν έλεγχοι αποδοτικότητας σε διάφορες μεταβλητές που επηρεάζουν τα ευρετικά. Εφόσον υπάρχει πλέον η αυτόματη πειραματική διαδικασία μέσο της οποίας χρειάζεται να γίνουν απλά αλλαγές στις παραμέτρους για να πάρουμε τα αποτελέσματα θα ήταν καλό να εκτελεστούν νέες σειρές πειραμάτων. Μια σειρά πειραμάτων μπορεί να συνεχίσει με πειράματα τα οποία αφορούν την αποδοτικότητα με βάση των αριθμό κανονικών βημάτων. Για τα πειράματα αυτά μπορεί να γίνει έλεγχος για 7, 10 και 15 κανονικά βήματα έτσι ώστε να βρεθεί ο κατάλληλος αριθμός βημάτων. Έπειτα μπορούν εκτελεστούν πειράματα που θα ασχοληθούν με την βελτιστοποίηση της φόρμουλας κάθε ευρετικού. Οι τιμές που τοποθετήθηκαν στις φόρμουλες έγιναν μέσω μικρού αριθμού πειραμάτων άρα μπορούν να βελτιωθούν μέσω μελλοντικών δοκιμών. Επίσης θα ήταν καλό να γίνουν δοκιμές με νέες τιμές στις φόρμουλες των κατωφλίων των ευρετικών οι οποίες υπολογίζουν σε ποια στιγμή πρέπει να διακοπεί η επαναληπτική διαδικασία. Τέλος μπορεί να γίνει σειρά πειραμάτων με νέα ευρετικά ή συνδυασμό του ευρετικού στόχων και κινήσεων για τη μεγιστοποιήσει των προβλημάτων που μπορούν να επιλυθούν.

Βιβλιογραφία

[1] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Torsten Schaub, "Multi-shot ASP solving with Clingo", *Theory and Practice of Logic Programming*, 19(1), 2019.

[2] Yiannis Dimopoulos, Martin Gebser, Patrick Luhne, Javier Romero, Torsten Shaub, "plasp 3: Towards Effective ASP Planning", *Theory and Practice of Logic Programming*, 19(3), 2019

[3] Martin Gebser, Roland Kaminski, Murat Knecht, Torsten Shaub, "plasp: A Prototype for PDDL-Based Planning in ASP", *International Conference on Logic Programming and Nonmonotonic Reasoning*, 2011.

[4] Martin Brain, Owen Cliffe, Marina De Vos, "A Pragmatic Programmer's Guide to Answer Set Programming", *Answer Set Programming*, 2009.

[5] Martin Gebser, Roland Kaminski, Benhamin Kaufmann, Marius Lindauer, Max Ostrowski, Javier Romero, Trosten Shaub Sven Thiele, Philipp Wanko, "Potassco User Guide", October 2017

[6] Martin Gebser, Roland Kaminski, Benhamin Kaufmann, Marius Lindauer, Max Ostrowski, Javier Romero, Trosten Shaub Sven Thiele,"Potassco User Guide", January 2019

[7] Malik Ghallab, Adele Howe, Craig Knoblock, Drew McDermott, Ashwin Ram, Manuela Veloso, Daniel Weld, David Wilkins, "PDDL - The Planning Domain Definition Language" - manual, October 1998

[8] Wolfgang Faber, Nicola Leone, Gerald Pfeifer, "Experimenting with Heuristics for Answer Set Programming", *17th Intern, Joint Conference on Artificial Intelligence* (*IJCAI)*, 2001.

[9] Ilkka Niemela, "Answer-Set Programmming: a Declarative Knowledge Representation Praadigm", *Lecture notes of ESSLLI*, 2001