Ατομική Διπλωματική Εργασία

ΕΞΕΡΕΥΝΗΣΗ ΙΕΡΑΡΧΙΚΩΝ ΜΕΘΟΔΩΝ ΣΤΟ ΠΛΑΙΣΙΟ ΕΝΙΣΧΥΤΙΚΗΣ ΜΑΘΗΣΗΣ ΠΡΑΚΤΟΡΩΝ

Μαρία Καλλή

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάιος 2015

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Εξερεύνηση Ιεραρχικών Μεθόδων στο πλαίσιο Ενισχυτικής Μάθησης Πρακτόρων

Μαρία Καλλή

Επιβλέπων Καθηγητής
Δρ. Χρίστος Χριστοδούλου

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

Μάιος 2015

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον επιβλέπων καθηγητή μου Δρ. Χρίστο Χριστοδούλου, που με εμπιστεύθηκε για την εκπόνηση της παρούσας διπλωματικής εργασίας αλλά και για την κατανόηση και εξαιρετική συνεργασία που είχαμε. Επίσης θα ήθελα να ευχαριστήσω τον διδακτορικό φοιτητή Βασίλη Βασιλειάδη για τη καθοδήγηση και τον συμβουλευτικό ρόλο του. Τέλος θα ήθελα να ευχαριστήσω τους γονείς μου, για την θερμή υποστήριξη και την ενθαρρυντική τους στάση καθ' όλη τη διάρκεια της εργασίας μου.

ΠΕΡΙΛΗΨΗ

Στόχος της διπλωματικής εργασίας είναι να εξεταστεί κατά πόσο οι ιεραρχικές μέθοδοι επιταχύνουν την μάθηση σε πολύπλοκα προβλήματα ενισχυτικής μάθησης. Συγκεκριμένα κλήθηκα να υλοποιήσω τον αλγόριθμο HexQ και να εξετάσω κατά πόσο λύνει με βέλτιστο τρόπο το «Πρόβλημα του Ταξί» (Dietterich, 2000). Παρόλο που δεν λήφθηκαν αποτελέσματα για τους λόγους που αναφέρω στο κεφάλαιο 6, ήμασταν πολύ κοντά στη λύση του προβλήματος.

Ο HexQ αποτελεί έναν πολύπλοκο αλγόριθμο, ο οποίος επιλύει σε λιγότερο υπολογιστικό χρόνο και συγχρόνως με μειωμένες απαιτήσεις μνήμης το μοντέλο του προβλήματος. Ούτως η άλλως τα προβλήματα ενισχυτικής μάθησης, πιθανόν να είναι κάποιες φορές πολύπλοκα προβλήματα, οπόταν η φύση του αλγορίθμου αυτού είναι ιδανική για αυτού του είδους τα προβλήματα.

Μαθαίνει την ιεραρχία των προβλημάτων, βρίσκοντας την βέλτιστη πολιτική, αυτόματα και η μάθηση γίνεται από κάτω προς τα πάνω. Άρα γίνεται μια προσπάθεια αποσύνθεσης και λύσης του προβλήματος με ιεραρχικό τρόπο, χωρίς γνώση του μοντέλου (προσπαθεί να μάθει αυτόματα). Ο αλγόριθμος σε περίπτωση που το πρόβλημα εμπλέκει περισσότερες από δυο μεταβλητές, αποκτά αναδρομικό χαρακτήρα. Διαχειρίζεται τα προβλήματα Μαρκοβιανής Διαδικασίας Απόφασης (ΜΔΑ) με ένα αφαιρετικό τρόπο απλοποίησης των καταστάσεων τους αλλά ταυτόχρονα συμπιέζει τα πολυδιάστατα προβλήματα ΜΔΑ. Πιστεύω ότι όντως ο αλγόριθμος αυτός οδηγεί σε βέλτιστες λύσεις στο πλαίσιο των προβλημάτων που εφαρμόζεται και αν ήμασταν σε θέση να εξάγαμε αποτελέσματα είμαι πεπεισμένη ότι θα ήταν καλύτερα από άλλους ιεραρχικούς αλγόριθμους ενισχυτικής μάθησης, γιατί φαίνεται απ’ την δομή και τις λειτουργίες

του αλγορίθμου πώς προσεγγίζει αυτού του είδους τα προβλήματα με ξεχωριστό τρόπο.

Περιεχόμενα

Λίστα Ακρωνυμίων

TP = Taxi Problem

ΜΔΑ = Μαρκοβιανή Διαδικασία Απόφασης

ΧΔ = Χρονική Διαφορά

ΤΝ = Τεχνητή Νοημοσύνη

IEM = Ιεραρχική Ενισχυτική Μάθηση

SCC = Strongly Connected Component

DAG = Directed Acyclic Graph

MER = Markov Equivalent Region

Sub-MDP=Υπο-ΜΔΑ

Κεφάλαιο 1

Εισαγωγή

Στόχος της διπλωματικής εργασίας είναι να εξεταστεί κατά πόσο οι ιεραρχικές μέθοδοι επιταχύνουν την μάθηση σε πολύπλοκα προβλήματα ενισχυτικής μάθησης. Ουσιαστικά, το αξιοσημείωτο χαρακτηριστικό των ιεραρχικών μεθόδων είναι ο τρόπος με τον οποίο εντοπίζονται επαναλαμβανόμενα, αμετάβλητα υπο-προβλήματα και πώς χρησιμοποιούνται και αναπαριστώνται με αφαιρετικό τρόπο στα πλαίσια της ενισχυτικής μάθησης, με στόχο την απλοποίηση του αρχικού μοντέλου προβλήματος και κατ' επέκταση την επίλυση του σε λιγότερο υπολογιστικό χρόνο και συγχρόνως με μειωμένες απαιτήσεις μνήμης.

Ο λόγος που καταπιαστήκαμε με την εξέταση των ιεραρχικών μεθόδων είναι διότι τα προβλήματα ενισχυτικής μάθησης, πιθανόν να είναι κάποιες φορές πολύπλοκα προβλήματα, προβλήματα με μεγάλο σύνολο μεταβλητών-καταστάσεων, που απαιτούν μεγάλο χώρο μνήμης για όλες τις πιθανές καταστάσεις και ενέργειες. Ο Bellman το 1961, αναφέρθηκε σ' αυτό το σημαντικό θέμα, το οποίο χαρακτήρισε και ως «η κατάρα της διαστατικότητας». Οι ιεραρχικές μέθοδοι στόχο έχουν να διασπάσουν το μεγάλο, αρχικό πρόβλημα σε υπο-προβλήματα και να λύσουν το κάθε υπο-πρόβλημα ξεχωριστά, ακολουθώντας οποιαδήποτε πολιτική, συνδυάζοντας τις λύσεις όλων των υπο-προβλημάτων που δημιουργήθηκαν για να

μας οδηγήσουν στη λύση του αρχικού προβλήματος. Αυτή η συμπαγής αναπαράσταση του προβλήματος που προκύπτει αποσυνθέτοντας το κάθε υπο-πρόβλημα σε μικρότερο, δημιουργώντας μια ιεραρχία (σειρά επιπέδων) και οι συνδυασμένες λύσεις των υπο-προβλημάτων, τείνουν συχνά σε μια ευκολότερη και πιο αποδοτική λύση του αρχικού προβλήματος.

Η ιεραρχική μέθοδος η οποία πρόκειται να εξεταστεί στα πλαίσια της εργασίας μου, είναι η Hierarchical EXit Q Function (HEXQ), που προτάθηκε από τον Hengst (2002), η οποία αυτόματα αποσυνθέτει το πρόβλημα και συνδυάζει τις λύσεις των υπο-προβλημάτων για να οδηγηθεί στην λύση του ολικού προβλήματος. Συνήθως οι πλείστοι προγραμματιστές που βρίσκονται στη διαδικασία μοντελοποίησης ενός προβλήματος με στόχο να το επιλύσουν μέσω μίας οποιαδήποτε ιεραρχικής μεθόδου, αποσυνθέτουν το πρόβλημα σε υπο-προβλήματα χειροκίνητα και δεν εκτελείται απ΄ το πρόγραμμα αυτόματα αυτή η διαδικασία αποσύνθεσης. Δεν υπάρχει αυτόματος τρόπος (χωρίς την εμπλοκή του προγραμματιστή) εύρεσης των υπο-προβλημάτων και διάσπασης του αρχικού προβλήματος χωρίς ιδιαίτερη δυσκολία. Και όμως ο ευρετικός αλγόριθμος HexQ χαρακτηρίζεται γι’ αυτό τον αυτοματισμό, την ταυτόχρονα αφαιρετική και συμπαγή αναπαράσταση των μεταβλητών του προβλήματος που δημιουργεί επεμβαίνοντας στο αρχικό μοντέλο. Δημιουργεί ένα τελικό σύνολο ιεραρχικών υπο-προβλημάτων τα οποία λύνει αναδρομικά με στόχο την επίλυση του αρχικού προβλήματος.

1.1 ΥΠΟΘΕΣΕΙΣ

Τα προβλήματα που λύνει ο αλγόριθμος HexQ είναι πεπερασμένα προβλήματα τύπου Μαρκοβιανής Διαδικασία Απόφασης (ΜΔΑ). Σε αυτού του είδους τα προβλήματα υπάρχει ένα σύνολο μεταβλητών, το οποίο χαρακτηρίζει το πρόβλημα. Γι' αυτό το λόγο, το σύνολο καταστάσεων λέγεται πολυδιάστατο. Οι καταστάσεις του προβλήματος διαφέρουν αναλόγως της μεταβλητής που χαρακτηρίζουν. Στο πρόβλημα το οποίο εφαρμόστηκε ο HexQ, το πρόβλημα του ταξί (Dietterich, 2000), αποτελείται από τρεις μεταβλητές, την θέση ταξί, την θέση επιβάτη και τον προορισμό. Άρα το σύνολο καταστάσεων στη συγκεκριμένη περίπτωση είναι τρισδιάστατο. Επίσης ο πράκτορας αρχικά θεωρούμε ότι δεν γνωρίζει το μοντέλο και πρέπει να το εξερευνήσει στα πλαίσια της μάθησης.

Η διάσπαση του προβλήματος απ' τον HexQ γίνεται μέσω επεξεργασίας της κάθε μίας μεταβλητή ξεχωριστά και σειριακά. Επίσης για να μπορεί να γίνει HexQ διάσπαση ενός ΜΔΑ πρέπει να τηρούνται οι πιο κάτω προϋποθέσεις :

- Το σύνολο καταστάσεων πρέπει να περιγράφεται από ένα σύνολο μεταβλητών που χαρακτηρίζουν το πρόβλημα.
- Πρέπει να υπάρχει διαφορά στη συχνότητα αλλαγής των μεταβλητών του προβλήματος (διότι ο αλγόριθμος στα αρχικά στάδια βασίζεται στη συχνότητα αλλαγής της κάθε μεταβλητής, ταξινομεί τις μεταβλητές με βάση τη συχνότητα αλλαγής τους και επεξεργάζεται κάθε μία ξεχωριστά με τη σειρά που ταξινομήθηκαν).
- Οι μεταβλητές που αλλάζουν συχνότερα τιμές πρέπει να διατηρούν τις ίδιες τιμές για να αντιπροσωπεύουν παρόμοια χαρακτηριστικά στα πλαίσια μεταβλητών που αλλάζουν πιο αργά.

- Οι πολιτικές μαθαίνονται μέσω των μεταβλητών που αλλάζουν συχνότερα τιμές, για να υπάρχει έλεγχος στη διάσχισης των υπο-χώρων (το σύνολο καταστάσεων των υπο-προβλημάτων).
- Τέλος, ο προγραμματιστής καλείται να ορίσει σωστά τις μεταβλητές του προβλήματος, διότι δεν έχει τόση σημασία το αν γνωρίζει το πώς θα διασπαστεί το πρόβλημα (διότι αυτό θα γίνει αυτόματα απ' τον HexQ) αλλά έχει σημασία ο ορισμός των μεταβλητών διότι θα επηρεάσουν την απόδοση του HexQ (θα δούμε στα επόμενα κεφάλαια γιατί οι μεταβλητές διαδραματίζουν τόσο σημαντικό ρόλο στον αλγόριθμο αυτό).

1.2 ΓΝΩΣΙΟΛΟΓΙΚΟ ΥΠΟΒΑΘΡΟ

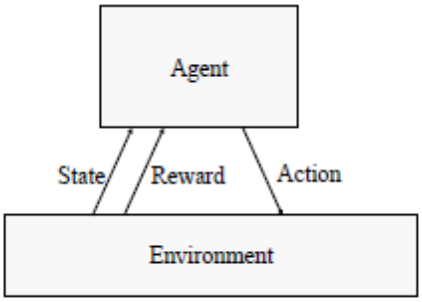
Η ενισχυτική μάθηση αποτελεί περιοχή της τεχνητής νοημοσύνης. Ασχολείται με δυναμικά προβλήματα όπως έλεγχο κάποιου ρομπότ, παιχνίδια στρατηγικής, ή προβλήματα σχεδιασμού (planning) κ.α. Καταπιάνεται με την μάθηση πρακτόρων, οι οποίοι πρέπει να ενεργούν ορθολογιστικά, δηλαδή μέσα από μια ακολουθία γεγονότων να αντιλαμβάνονται και να δρουν με τέτοιο τρόπο ούτως ώστε να μεγιστοποιήσουν την απόδοση τους. Πράκτορας ή εκπαιδευόμενος, είναι αυτός ο οποίος παίρνει τις αποφάσεις, είναι αυτός ο οποίος αλληλεπιδρά με το περιβάλλον, είναι αυτός ο οποίος προσπαθούμε να εκπαιδεύσουμε ούτως ώστε να μεγιστοποιηθεί ένα κριτήριο απόδοσης. Οτιδήποτε δεν μπορεί να μπορεί να ελέγξει ο πράκτορας αποτελεί κομμάτι του περιβάλλοντος στο οποίο κινείται, το οποίο αναπαρίσταται με καταστάσεις.

Στόχος αυτής της διαδικασίας είναι να μάθει ένα εκπαιδευόμενο, τον λεγόμενο πράκτορα, να συμπεριφέρεται όσο το δυνατό καλύτερα σ' ένα δυναμικό

περιβάλλον, στα πλαίσια ενός προβλήματος ενισχυτικής μάθησης. Αυτού του είδους τα προβλήματα είναι συνήθως πολύπλοκα και αποτελούνται από μεγάλα σύνολα μεταβλητών. Ο πράκτορας εμπλέκεται σε μια συνεχή διαδικασία δοκιμασίας-σφάλματος, αλληλεπιδρά με το περιβάλλον, το οποίο είναι άγνωστο προς αυτόν και πράττει αναλόγως. Ενισχύει δηλαδή, μια αλυσίδα αποφάσεων με τα εμπειρίας.

Το περιβάλλον μοντελοποιείται σαν σύνολο καταστάσεων και ενεργειών. Εκπέμπει ένα ενισχυτικό σήμα για κάθε ενέργεια που επιλέγει ο πράκτορας, το οποίο βοηθά ουσιαστικά τον εκπαιδευόμενο να αντιληφθεί εάν η ενέργεια που επέλεξε ήταν καλή ή όχι την δεδομένη στιγμή και ορίζει την επόμενη κατάσταση βάση της επιλεγμένης ενέργειας. Το περιβάλλον δεν ενημερώνει τον πράκτορα για το πώς θα πρέπει να κινηθεί μελλοντικά στο πλαίσιο του προβλήματος κατά τη διάρκεια του χρόνου, αλλά όπως προανέφερα του δίνει ως πληροφορία το ενισχυτικό σήμα το οποίο τον «προσανατολίζει» προς την κατεύθυνση της λύσης. Ουσιαστικά με την λέξη περιβάλλον εννοούμε τη συνάρτηση μετάβασης και την συνάρτηση αμοιβής.

Στόχος του πράκτορα είναι να επιλέξει μια ακολουθία ενεργειών, η οποία θα μεγιστοποιήσει ένα κριτήριο απόδοσης το οποίο ορίζουμε στο πρόβλημα ενισχυτικής μάθησης. Αυτό σημαίνει ότι ο πράκτορας εκπαιδεύεται με στόχο να βρει την βέλτιστη πολιτική για να μεγιστοποιήσει το ολικό κέρδος. Η πολιτική είναι μια στοχαστική συσχέτιση κατάστασης-ενέργειας. Όλα αυτά συμβαίνουν στα πλαίσια ενός επεισοδίου, όπου επεισόδιο ορίζουμε ως μια ακολουθία ενεργειών που επιλέγει ο εκπαιδευόμενος, μέχρι να φτάσει στο στόχο του. Το σχήμα (1.1) δείχνει την σχέση πράκτορα-περιβάλλοντος.



Σχήμα 1.1: Στο σχήμα παρατηρούμε την αλληλεπίδραση του πράκτορα με το περιβάλλον, το σήμα που εκπέμπει το περιβάλλον (σε μορφή αμοιβής) για κάθε ενέργεια που επιλέγει ο πράκτορας και την επόμενη κατάσταση που μεταβαίνει ο πράκτορας αναλόγως της επιλεγόμενης ενέργειας. Η εικόνα αυτή έχει παρθεί από τον Hengst (2002).

Κάπου εδώ πρέπει να αναφέρουμε ότι ο πράκτορας καλείται ανά πάσα στιγμή να επιλέξει μεταξύ εξερεύνησης και εκμετάλλευσης. Εκμετάλλευση συμβαίνει όταν ο πράκτορας επιλέγει σε μια δεδομένη στιγμή την ενέργεια η οποία θα του επιφέρει το μεγαλύτερο κέρδος από όλες τις πιθανές ενέργειες. Αυτή ορίζεται ως άπληστη επιλογή, διότι δίνει στον πράκτορα την μέγιστη αμοιβή κατά την τρέχουσα δεδομένη στιγμή, που σημαίνει ότι εκμεταλλεύεται την υπάρχουσα γνώση. Στην περίπτωση που δεν συμβεί αυτό, ο πράκτορας προχωρά στην εξερεύνηση νέων ενεργειών αφού επιλέγει μια μη-άπληστη επιλογή. Η εξερεύνηση πιθανόν να επιφέρει τη μέγιστη ολική αμοιβή και όχι την μέγιστη αμοιβή για μια δεδομένη στιγμή, όπως η εκμετάλλευση.

1.3 HEXQ

Προτάθηκε από τον Hengst το 2002 και μαθαίνει την ιεραρχία των προβλημάτων, βρίσκοντας την βέλτιστη πολιτική, αυτόματα. Η μάθηση γίνεται από κάτω προς τα

πάνω. Άρα γίνεται μια προσπάθεια αποσύνθεσης και λύσης του προβλήματος με ιεραρχικό τρόπο, χωρίς γνώση του μοντέλου (προσπαθεί να μάθει αυτόματα). Έχει κοινά στοιχεία με την προσέγγιση του MAXQ (Dietterich, 2000) όσο αφορά την αποσύνθεση της συνάρτησης αξίας και την φορά (από κάτω προς τα πάνω) που προσεγγίζει τα υπο-προβλήματα. Η αποσύνθεση του προβλήματος εξαρτάται σε μεγάλο βαθμό από την επιλογή των μεταβλητών του προβλήματος και από τη δομή του προβλήματος.

1.3.1 Γενική Περιγραφή του αλγορίθμου

Ο αλγόριθμος HexQ αποσυνθέτει προβλήματα ΜΔΑ και με βάση τις μεταβλητές του προβλήματος ψάχνει για συγκεκριμένες περιοχές (αποτελεί έννοια του αλγορίθμου που θα επεξηγηθεί στα επόμενα κεφάλαια). Όπως προαναφέρθηκε σε αυτό το κεφάλαιο, είναι εξαιρετικά σημαντικό στην απόδοση του αλγορίθμου η επιλογή των μεταβλητών του προβλήματος. Ο λόγος είναι διότι το $1^{ο}$ στάδιο του αλγόριθμου HexQ, είναι η ταξινόμηση των μεταβλητών του προβλήματος με βάση τη συχνότητα αλλαγής τους. Αυτή η ταξινόμηση θα ορίσει τη σειρά επεξεργασίας της κάθε μεταβλητής, αφού ο συγκεκριμένος αλγόριθμος επεξεργάζεται μία μεταβλητή κάθε φορά.

Αρχικά εκτελεί μια τυχαία εξερεύνηση για να παρατηρήσει τη συχνότητα αλλαγής των μεταβλητών, τη συνάρτηση μετάβασης και τη συνάρτηση αξίας διότι το μοντέλο στον αλγόριθμο αυτό δεν είναι γνωστό εξ’ αρχής, το ανακαλύπτει μέσω τυχαίας εξερεύνησης πριν αρχίσει η αποσύνθεση του προβλήματος σε υπο-προβλήματα, όπου τα υπο-προβλήματα αποτελούν μικρότερα προβλήματα ΜΔΑ,

υπο-ΜΔΑ (sub-MDP, Markov Decision Process) όπως αναφέρονται στην διδακτορική εργασία του Hengst (2002). Γι αυτό και όποτε αναφέρεται στην παρούσα εργασία η λέξη sub-MDP εννοείται ένα υπο-πρόβλημα τύπου ΜΔΑ. Όσες είναι οι μεταβλητές του προβλήματος, τόσα είναι και τα επίπεδα ιεραρχίας που δημιουργούνται.

Για κάθε μεταβλητή που χειρίζεται (δηλαδή για κάθε επίπεδο), ο αλγόριθμος προσπαθεί να εντοπίσει περιοχές που περιέχουν sub-MDPs. Επίσης προσπαθεί να εντοπίσει «εξόδους», οι οποίες παράγονται από μη-αναμενόμενες μεταβάσεις ή από μεταβάσεις που προκαλούν μη-αναμενόμενες ανταμοιβές. Όταν εκτελεστεί έξοδος σημαίνει ότι ο εκπαιδευόμενος έχει εξέλθει από μια περιοχή. Άρα ο εκπαιδευόμενος εξερευνεί τις καταστάσεις της περιοχής που βρίσκεται και με πιθανότητα 1 εκτελεί σε ένα βάθος χρόνου μετάβαση εξόδου. Η έξοδος είναι ο μοναδικός τρόπος να εξέλθει ο εκπαιδευόμενος από την περιοχή που βρίσκεται.

Σε κάθε περιοχή υπάρχουν πολλαπλά sub-MDPs τα οποία τερματίζονται όταν ο πράκτορας επιλέξει μία από τις εξόδους της περιοχής. Σε κάθε sub-MDP αντιστοιχεί και μία έξοδος της περιοχής. Αυτός είναι ο υπο-στόχος του κάθε sub-MDP. Κάθε sub-MDP περιέχει ως στόχο του μία κατάσταση εξόδου και μόνο όταν ο πράκτορας εξέλθει της περιοχής από την συγκεκριμένη έξοδο μπορεί να τερματίσει το συγκεκριμένο υπο-πρόβλημα και να ικανοποιηθεί ο υπο-στόχος. Έτσι στα πλαίσια της εξερεύνησης κάθε περιοχής, οι πολιτικές που ακολουθούνται αποθηκεύονται (περισσότερες λεπτομέρειες στα επόμενα κεφάλαια για αναλυτική περιγραφή του αλγορίθμου).

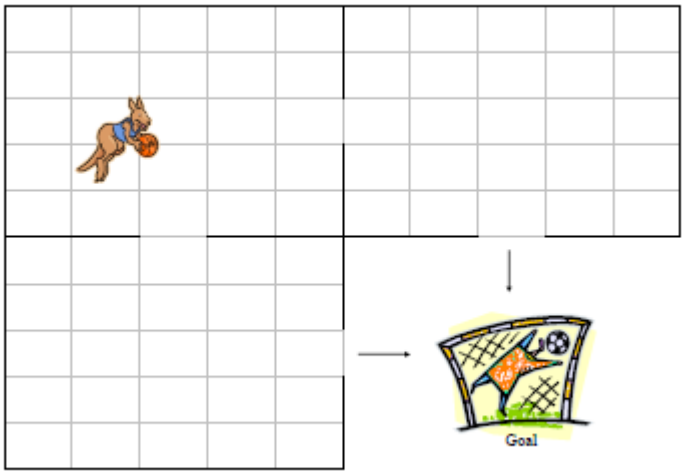
Όταν γίνει έξοδος από μια περιοχή τότε ο αλγόριθμος δημιουργεί ένα νέο αφαιρετικό ΜΔΑ, το οποίο αντιστοιχεί στην επόμενη μεταβλητή, δηλαδή στο αμέσως ψηλότερο ιεραρχικό επίπεδο. Οι καταστάσεις αυτού του νέου ΜΔΑ είναι αφαιρετικές. Οι νέες καταστάσεις στο σύνολο τους αποτελούν το καρτεσιανό γινόμενο του αριθμού των περιοχών του προηγούμενου επιπέδου επί το εύρος τιμών της νέας μεταβλητής. Οι αφαιρετικές ενέργειες αυτού του επιπέδου είναι οι αποθηκευμένες πολιτικές του προηγούμενου επιπέδου (αφαιρετικές ενέργειες σημαίνει ότι σε ένα βάθος χρόνου εκτελούνται κάποια βήματα μέχρι να φτάσει ο πράκτορας στην έξοδο).

Οι πιο πάνω διαδικασίες εκτελούνται επαναλαμβανόμενα για κάθε μεταβλητή μέχρι να παραμείνει η τελευταία (η μεταβλητή που έχει τη μικρότερη συχνότητα αλλαγής) όπου τυγχάνει διαφορετικού χειρισμού. Στο τελευταίο δηλαδή επίπεδο ιεραρχίας, το ανώτατο, υπάρχει ένα και μοναδικό υπο-πρόβλημα, το οποίο όταν λυθεί, λύνεται το ολικό ΜΔΑ.

Η συνάρτηση αξίας του αλγορίθμου, συντίθενται από τις συσσωρευμένες αμοιβές του πράκτορα σε μια περιοχή μέχρι να φτάσει σε έξοδο συν την αξία της επόμενης αφαιρετικής ενέργειας.

Θα επεξηγηθεί το απλό παράδειγμα του σχήματος (1.2) για να δώσουμε μια πρώτη ιδέα στο τι θα μπορούσε να ήταν ένα πρόβλημα ενισχυτικής μάθησης, πώς αντιμετωπίζεται από τον αλγόριθμο HexQ, τι σημαίνουν σε πρακτική εφαρμογή οι έννοιες που επεξηγήθηκαν στο υποκεφάλαιο 1.3.1, πώς αποσυντίθεται σε υπο-προβλήματα το αρχικό πρόβλημα ΜΔΑ, πώς οι λύσεις των υπο-προβλημάτων συνδυάζονται για την επίλυση του αρχικού προβλήματος, τι ρόλο έχουν οι πολιτικές του κάθε υπο-προβλήματος στη λύση του προβλήματος και γιατί είναι

σημαντικές κ.α. Δεν μας ενδιαφέρει το πρόβλημα αυτό καθ' αυτό παρά μόνο κάποιες σημαντικές παρατηρήσεις που προκύπτουν απ' αυτό το απλό παράδειγμα, με στόχο την εισαγωγή του αναγνώστη στα προβλήματα ενισχυτικής μάθησης.



Σχήμα 1.2: Απεικονίζεται το δισδιάστατο πρόβλημα του λαβύρινθου, ένα σχετικά απλό πρόβλημα στο οποίο υπάρχουν τρία δωμάτια συνδεδεμένα μέσω διαδρόμων (κενά). Το κάθε δωμάτιο αποτελείται από εικοσιπέντε θέσεις το καθένα και ο λαγός (πράκτορας) πρέπει να οδηγηθεί στον στόχο (μέσω ενός εκ των δυο διαδρόμων) όσο το δυνατό γρηγορότερα. Ο λαγός έχει την δυνατότητα να μετακινηθεί στις τέσσερις κατευθύνσεις (Βορρά, Νότο, Ανατολή, Δύση) και γνωρίζει σε ποια θέση και σε ποιο δωμάτιο βρίσκεται ανά πάσα στιγμή. Η εικόνα αυτή έχει παρθεί από τον Hengst (2002).

Ο πράκτορας σε οποιοδήποτε πρόβλημα ενισχυτικής μάθησης στόχο έχει να βρει μια βέλτιστη συνάρτηση αξίας. Στη συγκεκριμένη περίπτωση ο πράκτορας αναπαριστάται από τον λαγό στο σχήμα (1.2) και στόχο έχει να ελαχιστοποιήσει τα βήματα του ως προς το στόχο. Οπόταν μια πρώτη προσέγγιση του προβλήματος θα ήταν (άσχετα με τη συνάρτηση αξίας του συγκεκριμένου προβλήματος) ο

πράκτορας να επιλέγει κάθε φορά το κελί το οποίο θα του δίνει την μικρότερη αξία, αφού στόχος του είναι η ελαχιστοποίηση των βημάτων του μέχρι το στόχο.

Το πρόβλημα αποτελείται από δυο μεταβλητές (προϋπόθεση 1 υποκεφάλαιο 1.1), τη *θέση εντός δωματίου* και τον *αριθμό δωματίου*, συνεπώς θα υπάρξουν δυο επίπεδα ιεραρχίας. Παρατηρούμε ότι η μεταβλητή *θέση εντός δωματίου* λογικά αλλάζει τιμές συχνότερα από την μεταβλητή *αριθμός δωματίου* (προϋπόθεση 2 υποκεφάλαιο 1.1).

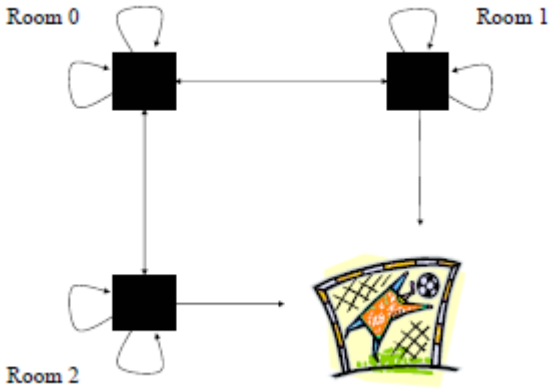
Επίσης η διάσχιση του κάθε δωματίου είναι πανομοιότυπη, αφού η συνάρτηση μετάβασης και η συνάρτηση αξίας είναι η ίδια για κάθε δωμάτιο. Το κάθε δωμάτιο αποτελεί περιοχή και αναγνωρίζουμε κοινά sub-MDPs στις περιοχές, αφού ένα sub-MDP στη συγκεκριμένη περίπτωση είναι το πώς ο λαγός θα εξέλθει του δωματίου μέσω ενός απ' τους διαδρόμους (ο κάθε διάδρομος αποτελεί έξοδο. Υπάρχει η έννοια υπο-πρόβλημα και όχι πρόβλημα διότι δεν λύνεται το ολικό πρόβλημα αν ο λαγός εξέλθει απλά του δωματίου μέσω του διαδρόμου.

Επίσης η τιμή που θα λάβει ο πράκτορας για να φτάσει στο στόχο μετά απ' την έξοδο του δωματίου διαφέρει απ' την τιμή που θα λάβει μέχρι να εξέλθει του δωματίου. Οπόταν η συνάρτηση αξίας εντός του δωματίου διαφέρει από την συνάρτηση αξίας μετέπειτα. Θα μπορούσε να επαναχρησιμοποιηθεί η συνάρτηση αξίας εντός του δωματίου και για διάσχιση άλλου δωματίου αφού τα υπο-προβλήματα είναι κοινά. Αξιοσημείωτο είναι το γεγονός ότι στην παρουσία κοινών υπο-προβλημάτων όπως στο συγκεκριμένο παράδειγμα, δεν σημαίνει ότι ο

πράκτορας δεν θα εξερευνήσει κοινές περιοχές (που έχουν την ίδια συνάρτηση μετάβασης και συνάρτηση αξίας).

Επίσης επισημαίνεται η προϋπόθεση τρία που ορίσαμε στο υποκεφάλαιο 1.1. Στη συγκεκριμένη περίπτωση έχουμε τη μεταβλητή *θέση εντός δωματίου* η οποία χαρακτηρίζει τη μεταβλητή που αλλάζει συχνότερα τιμή και τη μεταβλητή *αριθμός δωματίου* που χαρακτηρίζει τη μεταβλητή που αλλάζει πιο αργά τιμή.

Όταν ο πράκτορας μάθει το πώς να ξεφεύγει απ' τις περιοχές (στη συγκεκριμένη περίπτωση τα τρία δωμάτια) με κάποια πολιτική, τότε το πρόβλημα γίνεται αφαιρετικό (μαύροι κύκλοι-σχήμα (1.3)) καθώς αλλάζει η μεταβλητή που επεξεργάζεται ο HexQ, αλλάζει το επίπεδο ιεραρχίας (ανεβαίνει επίπεδο), και τώρα επικεντρώνεται στο πώς συνδέονται τα υπο-προβλήματα.



Σχήμα 1.3: Αφαιρετικό μοντέλο που προκύπτει αφότου ο πράκτορας μάθει τις πολιτικές που τον οδηγούν εκτός του κάθε δωματίου (άρα δεν απεικονίζονται οι λεπτομέρειες εντός του δωματίου). Τα βέλη είναι αφαιρετικές ενέργειες που οδηγούν απ' το ένα δωμάτιο στο άλλο και στο στόχο.

Αυτή η εικόνα αποτελεί την αφαιρετική αναπαράσταση ενός προβλήματος δυο μεταβλητών. Η εικόνα αυτή έχει παρθεί από τον Hengst (2002).

Κεφάλαιο 2

Προκαταρκτικές έννοιες και εξισώσεις

Στα πλαίσια της μάθησης ο πράκτορας επιλέγει σειρά ενεργειών και συγχρόνως εκτελεί σειρά βημάτων στο χρόνο. Αν η πιθανότητα της επόμενης κατάστασης εξαρτάται πλήρως από την προηγούμενη κατάσταση και ενέργεια, τότε λέμε ότι ικανοποιείται η ιδιότητα Markov όπως φαίνεται στην εξίσωση (2.1). Αυτή η συνθήκη ισχύει και για τις αμοιβές, δηλαδή αν η επόμενη αμοιβή εξαρτάται μόνο από την προηγούμενη κατάσταση και ενέργεια τότε ικανοποιεί την ιδιότητα Markov.

$$Pr\{s_{t+1} = s' | s_t, a_t\} \tag{2.1}$$

2.1 ΜΔΑ

Υιοθέτησε το όνομα αυτό από τον Andrey Markov (1856-1922) ο οποίος εισήγαγε το ομώνυμο μοντέλο. Μια Μαρκοβιανή Διαδικασία Απόφασης αποτελεί μια πλειάδα < S,A,T,R > όπου S πεπερασμένο σύνολο καταστάσεων, A πεπερασμένο σύνολο ενεργειών, T συνάρτηση μετάβασης T : S×A×S→[0,1] και R συνάρτηση αξίας R : S×A×S→R. Η συνάρτηση μετάβασης και η συνάρτηση αξίας ορίζουν το μοντέλο Μαρκοβιανής Διαδικασίας Απόφασης.

Η διάσταση ενός ΜΔΑ ορίζεται από τον αριθμό των μεταβλητών που χαρακτηρίζει το πρόβλημα. Έστω d διαφορετικές μεταβλητές, τότε το ΜΔΑ είναι διάστασης d.

2.1.1 Μοντέλο

Με την έννοια μοντέλο εννοούμε τη συνάρτηση μεταβάσεων και τη συνάρτηση αξίας. Σε προβλήματα δυναμικού προγραμματισμού το μοντέλο είναι γνωστό εξ' αρχής ενώ σε προβλήματα ενισχυτικής μάθησης το μοντέλο δεν είναι γνωστό εξ' αρχής αλλά ο πράκτορας το εξερευνά στα πλαίσια της εκπαίδευσης.

2.1.2 Πολιτική

Είναι μία αντιστοίχιση μεταξύ καταστάσεων και πιθανών ενεργειών. Υπάρχουν περιπτώσεις όπου για μια κατάσταση s επιλέγεται ενέργεια a με πιθανότητα π(s,a). Αυτό σημαίνει ότι το μοντέλο είναι στοχαστικό. Σε αντίθεση με ένα ντετερμινιστικό μοντέλο όπου για όλες τις καταστάσεις s και για όλες τις ενέργειες a επιλέγεται ενέργεια a από κατάσταση s με πιθανότητα π(s,a)=1.

2.1.3 Συνάρτηση Κατάστασης-Αξίας

Η συνάρτηση κατάστασης-αξίας που χρησιμοποιείται στον αλγόριθμο HexQ απεικονίζεται στην εξίσωση (2.2) όπου αποτελεί το άθροισμα των εκπτωτικών αναμενόμενων μελλοντικών αξιών. Αλλά για ένα ΜΔΑ με πολιτική π μπορεί να γραφτεί όπως παρατηρούμε στην εξίσωση (2.3), ως η αναμενόμενη ανταμοιβή της επόμενης κατάστασης και η εκπτωτική αναμενόμενη αξία αυτής. Σαφώς αν ψάχνουμε την ενέργεια που θα μας δώσει την βέλτιστη αξία από κατάσταση s, από ΜΔΑ m, τότε προκύπτει η εξίσωση (2.4).

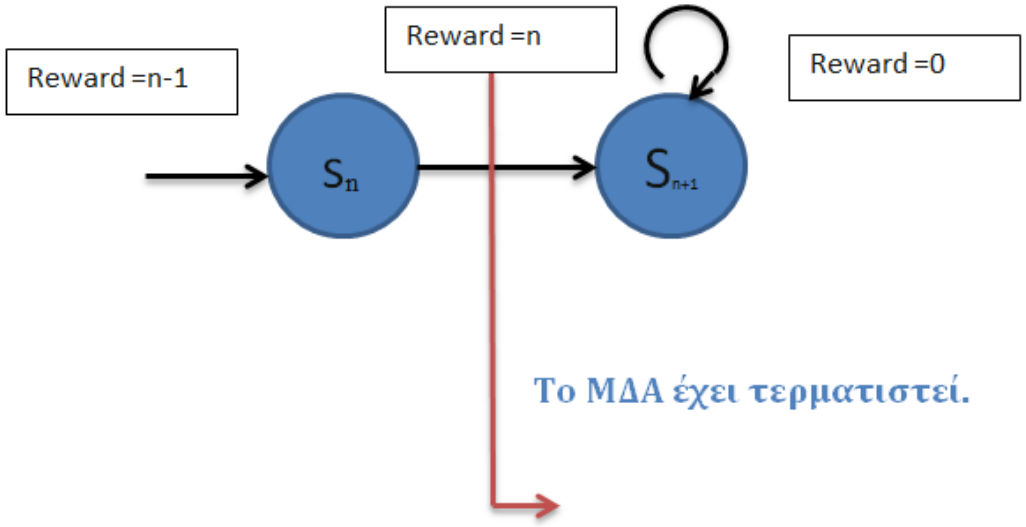
$$V_m^{\pi}(s) = E\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} .... | s_t = s, \pi\} \tag{2.2}$$

$$V_m^{\pi}(s) = \sum_{s'} T_{ss'}^{\pi(s)} [R_{ss'}^{\pi(s)} + \gamma V_m^{\pi}(s')] \tag{2.3}$$

$$V_m^{*}(s) = \max_a \sum_{s'} T_{ss'}^{a} [R_{ss'}^{a} + \gamma V_m^{*}(s')] \tag{2.4}$$

2.1.4 Επεισοδιακό ΜΔΑ

Επεισοδιακό ΜΔΑ σημαίνει ότι σε ένα βάθος χρόνου το ΜΔΑ θα τερματιστεί με πιθανότητα ένα. Ο ρυθμός έκπτωσης σε επεισοδιακά ΜΔΑ είναι ίσος με ένα. Δεν είναι όλα τα ΜΔΑ επεισοδιακά αφού υπάρχουν και ΜΔΑ που διαρκούν απείρως. Οπόταν για επεισοδιακά ΜΔΑ υποθέτουμε ότι κατά τον τερματισμό του προβλήματος, γίνεται μετάβαση σε μια κατάσταση τερματισμού (absorbing state) όπως φαίνεται στο σχήμα (2.1), όπου δεν λαμβάνονται πλέον αμοιβές απ' το περιβάλλον αφού το πρόβλημα έχει τερματιστεί. Ο ρυθμός έκπτωσης σε μη-επεισοδιακά ΜΔΑ κυμαίνεται από μηδέν έως ένα.



Σχήμα 2.1: Στο σχήμα φαίνεται η μετάβαση από την τελευταία κατάσταση του προβλήματος S_n, στην κατάσταση τερματισμού S_{n+1}, όπου απ' την κόκκινη κατακόρυφη γραμμή και μετά το πρόβλημα έχει τερματιστεί και δεν λαμβάνονται απ' το περιβάλλον άλλες αμοιβές.

2.1.5 Συνάρτηση Ενέργειας-Αξίας

Η συνάρτηση Ενέργειας-Αξίας $Q^{\pi}(s, a)$ όπως φαίνεται στην εξίσωση (2.5) είναι αναμενόμενη επιστροφή που θα δώσει η κατάσταση s με βάση μια ενέργεια a ακολουθώντας μια πολιτική π.

$$Q_m^{\pi}(s,a) = E\{r_{t+1} + \gamma V_m^{\pi}(s_{t+1}) | s_t = s, a_t = a\} \tag{2.5}$$

Αν όμως αντικαταστήσουμε την εξίσωση (2.5) με την εξίσωση (2.3) τότε προκύπτει η εξίσωση (2.6).

$$Q_m^{\pi}(s,a) = \sum_{s'} T_{ss'}^{a}[R_{ss'}^{a} + \gamma Q_m^{\pi}(s', \pi(s'))]$$

(2.6)

Και αν ψάχνουμε την βέλτιστη τιμή Q τότε προκύπτει η εξίσωση (2.7).

$$Q_m^{*}(s,a) = \sum_{s'} T_{ss'}^{a}[R_{ss'}^{a} + \gamma \max_{a'} Q_m^{*}(s', a')]$$

(2.7)

Όπως αναφέραμε πιο πάνω, σε περιπτώσεις όπου η συνάρτηση μεταβάσεων και η συνάρτηση ανταμοιβής είναι γνωστά, σημαίνει ότι το μοντέλο είναι γνωστό εκ των προτέρων. Άρα για να λυθεί ένα ΜΔΑ κάτω απ' αυτές τις συνθήκες θα μπορούσε να χρησιμοποιηθεί ο αλγόριθμος Value Iteration που φαίνεται στο σχήμα (2.2).

```
function ValueIteration( MDP⟨S, A, T, R⟩, γ )
    initialise Q(s, a) ← 0
    repeat until △ < small positive number
        △ ← 0
        for each state s ∈ S
            for each action a ∈ A
                q ← Q(s, a)
                Q(s, a) ← Σ_{s'} T^a_{ss'} [R^a_{ss'} + γV(s')]
                △ ← max(△, |q − Q(s, a)|)
    end repeat
    return Q(s, a)
end ValueInteration
```

Σχήμα 2.2: Ο αλγόριθμος ValueIteration που χρησιμοποιείται σε περιπτώσεις όπου το μοντέλο είναι γνωστό εξ' αρχής, γι' αυτό όπως φαίνεται στα ορίσματα της συνάρτησης υπάρχει το ΜΔΑ. Ο αλγόριθμος έχει παρθεί από τον Hengst (2002) και είναι εξαιρετικά σημαντικός για την λύση του προβλήματος.

Ενώ σε περιπτώσεις όπου το μοντέλο δεν είναι γνωστό χρησιμοποιούμε αλγόριθμους όπως ο γνωστός Q-Learning (Watkins, 1989), ο οποίος ταυτόχρονα εξερευνά το περιβάλλον και μαθαίνει την βέλτιστη συνάρτηση αξίας V. Ο αλγόριθμος Q-Learning υπάρχει στο σχήμα (2.3).

function Q-Learning
  initialise $Q(s,a) \leftarrow 0$ and set the learning rate β
  initialise $s \leftarrow$ initial state
  repeat until termination
    choose action a using exploration policy derived from Q
    take action a, observe r, s'
    $Q(s,a) \leftarrow (1-\beta)Q(s,a) + \beta[r + \gamma \max_{a'} Q(s',a')]$
    $s \leftarrow s'$
end Q-Learning

Σχήμα 2.3: Ο αλγόριθμος Q-Learning μαθαίνει το μοντέλο ταυτόχρονα με την εξερεύνηση και υιοθετήθηκε και παραλλάχθηκε από πολλούς, όπως οι Sutton και Barto (1998). Ο αλγόριθμος έχει παρθεί από τον Hengst (2002), δεν χρησιμοποιήθηκε στα πλαίσια της εργασίας.

2.1.6 Αφαιρετικές Ενέργειες σε sub-MDPs

Η έννοια αφαιρετική ενέργεια χρησιμοποιήθηκε στο προηγούμενο κεφάλαιο στο παράδειγμα του υποκεφαλαίου 1.3 για να δείξουμε το πώς ένα ΜΔΑ από ένα σημείο και έπειτα μετατρέπεται σε ένα αφαιρετικό μοντέλο και έτσι αναφερθήκαμε σε αφαιρετικές ενέργειες. Οι αφαιρετικές ενέργειες είναι εξαιρετικά σημαντικές στην ιεραρχική ενισχυτική μάθηση (IEM) αφού μία αφαιρετική ενέργεια οδηγεί στην λύση ενός υπο-προβλήματος (sub-MDP). Αποτελεί αφαιρετική ενέργεια και όχι απλά ενέργεια, διότι μία αφαιρετική ενέργεια είναι το σύνολο κάποιων ενεργειών, είναι σειρά από επιλεγμένες ενέργειες του πράκτορα που θα οδηγήσουν στον τερματισμό ενός υπο-προβλήματος. Ουσιαστικά όταν ο πράκτορας βρίσκεται σε ένα επίπεδο ιεραρχίας στο όποιο καλείται να επιλέξει μια απ' τις διαθέσιμες αφαιρετικές ενέργειες σημαίνει ότι σίγουρα δεν βρίσκεται στο κατώτατο επίπεδο, διότι εκεί υπάρχουν διαθέσιμες μόνο *primitive* ενέργειες.

Όπως είδαμε στο σχήμα 1.3, με μαύρους κύκλους απεικονίζονται οι πολιτικές του προηγούμενου επιπέδου, δηλαδή η διάσχιση του πράκτορα (η σειρά κινήσεων που εκτέλεσε) εντός του κάθε δωματίου. Όταν ο πράκτορας εξέλθει του δωματίου τότε από primitive ενέργειες υπάρχουν αφαιρετικές ενέργειες διότι το μοντέλο έχει σμικρυνθεί και έχει γίνει πιο αφαιρετικό, με λιγότερη λεπτομέρεια όπως βλέπουμε στο σχήμα 1.3.

Μια αφαιρετική ενέργεια στο παράδειγμα του υποκεφαλαίου 1.3 θα μπορούσε να ήταν «φύγε από τον ΒΔ δωμάτιο απ' τον ανατολικό διάδρομο». Άρα

αντιλαμβανόμαστε ότι για να επιτευχθεί αυτή η ενέργεια χρειάζεται ο πράκτορας να εκτελέσει περισσότερα από ένα βήματα εντός του ΒΔ δωματίου για να ξεφύγει απ’ την συγκεκριμένη περιοχή και να τερματιστεί αυτή η αφαιρετική ενέργεια. Το πώς θα το καταφέρει αυτό ο πράκτορας δεν μας ενδιαφέρει γι’ αυτό και οι κύκλοι απεικονίζονται με μαύρο χρώμα στο αφαιρετικό μοντέλο του σχήματος 1.3.

Κεφάλαιο 3

Αναδρομή στις τεχνικές της Ενισχυτικής Μάθησης.

Σ’ αυτό το κεφάλαιο γίνεται μια εισαγωγή και γενική περιγραφή των μεθόδων προσέγγισης συναρτήσεων, ιεραρχικών μεθόδων και μεθόδων ενισχυτικής μάθησης, με στόχο την κατανόηση της συμβολής των μεθόδων αυτών στον τομέα της ενισχυτικής μάθησης και γενικότερα στο πλαίσιο των ιεραρχικών προσεγγίσεων.

3.1 Προσέγγιση Συναρτήσεων

Οι τεχνικές προσέγγισης συναρτήσεων (Sutton και Barto, 1998), χρησιμοποιούνται για τη συμπαγή αναπαράσταση της συνάρτησης αξίας. Η συνάρτηση αξίας είναι ένας δισδιάστατος πίνακας, όπου στη μία διάσταση αντικατοπτρίζεται το σύνολο καταστάσεων και στην άλλη διάσταση το σύνολο των ενεργειών. Φυσικά παρουσιάζεται πρόβλημα μνήμης για προβλήματα με μεγάλο σύνολο καταστάσεων και ενεργειών (όπως αυτά που εμφανίζονται συχνά στα πλαίσια της ενισχυτικής μάθησης) αφού χρειαζόμαστε τεράστιους πίνακες και πολύ υπολογιστικό χρόνο μέχρι να γεμίσουν οι πίνακες. Η γενίκευση είναι ο μόνος τρόπος μέσα από ένα υποσύνολο καταστάσεων να βρεθεί μια καλή προσεγγιστική λύση χωρίς τα προαναφερθέντα προβλήματα, που να αντικατοπτρίζει όλο το πρόβλημα.

Αυτή η λύση δεν είναι μπορεί να χρησιμοποιηθεί μόνη της σε προβλήματα ενισχυτικής μάθησης, όπου το μοντέλο δεν είναι γνωστό εκ των προτέρων, άρα δεν γνωρίζουμε αυτό το υποσύνολο καταστάσεων που θα μας δώσει μια καλή προσεγγιστική λύση, χωρίς προβλήματα χρόνου ή/και μνήμης. Έτσι η γενίκευση από προηγούμενες εμπειρίες δεν λύνουν το πρόβλημα.

Το πρόβλημα λύνεται όμως, με τον συνδυασμό ενισχυτικής μάθησης και μεθόδων γενίκευσης. Αυτή η λεγόμενη γενίκευση αποτελεί και την προσεγγιστική συνάρτηση. Η προσεγγιστική συνάρτηση προσπαθεί μέσω παραδειγμάτων που παράγουν την βέλτιστη λύση, να δημιουργήσει μια συνάρτηση που θα αντικατοπτρίζει εξ’ ολοκλήρου το πρόβλημα. Άρα εφ’ όσον παίρνει παραδείγματα και παράγει αποτελέσματα βάσει αυτών, ανήκει στην επιβλεπόμενη μάθηση.

Το φλέγον θέμα όμως είναι το πώς αυτές οι τεχνικές προσέγγισης συναρτήσεων θα συμβαδίσουν και θα προσαρμοστούν με τις ιδιαιτερότητες και τα χαρακτηριστικά της ενισχυτικής μάθησης. Στην ενισχυτική μάθηση ο πράκτορας μαθαίνει συνεχώς, ανανεώνει συνεχώς αυτή την συνάρτηση, χρησιμοποιεί περισσότερες από μια πολιτικές, εξερευνά και εκμεταλλεύεται… Αυτά και άλλα πολλά ζητήματα πρέπει να ληφθούν υπόψη για να μπορούν αυτές οι τεχνικές να αντιμετωπίζουν προβλήματα ενισχυτικής μάθησης με επιτυχία.

3.2 Ιεραρχικές Μέθοδοι

Χρησιμοποιούνται σε δομημένα προβλήματα με στόχο την διάσπαση τους σε υπο-προβλήματα. Χρησιμοποιούν πολιτικές των υπο-προβλημάτων και όχι μια μεμονωμένη πολιτική προκαθορισμένη εξ' αρχής. «Μεσολαβούν πολλαπλά επίπεδα αποφάσεων όπου συνολικά λύνουν το πρόβλημα» (Hengst, 2002). Αυτή η επαναχρησιμοποίηση ωφελεί και σε υπολογιστικό χρόνο, αφού οδηγούμαστε στη λύση γρηγορότερα. Οι ιεραρχικές μέθοδοι διαδραματίζουν καθοριστικό ρόλο στον τομέα της ενισχυτικής μάθησης αφού υπάρχουν αλγόριθμοι που ψάχνουν την ιεραρχία του προβλήματος, αποσυνθέτουν το αρχικό πρόβλημα σε υπο-προβλήματα αυτόματα (όπως ο αλγόριθμος HexQ (Hengst, 2002)) και μη (όπως ο αλγόριθμος MaxQ (Dietterich, 2000)).

3.3 Μέθοδοι Ενισχυτικής Μάθησης

Ακολουθούν στα επόμενα υποκεφάλαια τρεις βασικές κατηγορίες μεθόδων ενισχυτικής μάθησης.

3.3.1 Μέθοδοι Monte Carlo

Οι μέθοδοι Monte Carlo (Michie και Chambers, 1968), δεν χρειάζονται πλήρη γνώση του περιβάλλοντος (μοντέλο) σε αντίθεση με τις μεθόδους δυναμικού προγραμματισμού, αλλά μαθαίνουν μέσω προσομοιωμένης εμπειρίας με τ' επεισοδίων. Μαθαίνουν κατ' ευθείαν απ' το περιβάλλον χρησιμοποιώντας ένα αλγόριθμο εντός-πολιτικής (παρόλο που ακόμη δεν έχει επιτευχθεί βέλτιστη συμπεριφορά) όπου βελτιώνει την πολιτική ταυτόχρονα με την εξερεύνηση ή χρησιμοποιείται ένας αλγόριθμος εκτός-πολιτικής. Μέσω δειγματοληψίας ή προσέγγισης των καταστάσεων διατηρούν ένα σύνολο καταστάσεων. Προσπαθούν

να λύσουν προβλήματα ενισχυτικής μάθησης και να εκτιμήσουν τη συνάρτηση αξίας, υπολογίζοντας τον μέσο όρο των επισκεφθέντων παραδειγμάτων με βάση μια κατάσταση .

3.3.2 Μέθοδοι Δυναμικού Προγραμματισμού

Οι μέθοδοι δυναμικού προγραμματισμού (Bellman, 1957) γνωρίζοντας εξ' ολοκλήρου το περιβάλλον, ένα πλήρες μοντέλο, ένα πεπερασμένο ΜΔΑ, μπορούν να υπολογίσουν τη βέλτιστη πολιτική ενεργώντας σε όλο το σύνολο καταστάσεων και ενεργειών. Γενικότερα αυτές οι μέθοδοι χρησιμοποιούν συναρτήσεις αξίας κι έτσι οργανώνουν τον χώρο αναζήτησης για εύρεση καλών πολιτικών. Υπάρχουν δυο μέθοδοι που το κάνουν αυτό: Η αξιολόγηση πολιτικής και βελτίωση πολιτικής. Η $1^{η}$ προσπαθεί να υπολογίσει, να προβλέψει την συνάρτηση κατάστασης-αξίας V(s) ενώ η $2^{η}$ προσπαθεί να βελτιώσει την αρχική πολιτική για επιστροφή καλύτερου αναμενόμενου κέρδους. Δεν χρησιμοποιούνται ευρέως στο πλαίσιο της ενισχυτικής μάθησης διότι είναι υπολογιστικά ακριβές μέθοδοι και προαπαιτούν τη γνώση μοντέλου (ενώ δεν είναι πάντα διαθέσιμο). Εφαρμόζονται ευρέως σε συνεχή προβλήματα καταστάσεων και ενεργειών (continuous time and space).

3.3.3 Μέθοδοι Χρονικών Διαφορών

Οι μέθοδοι χρονικών διαφορών είναι συνδυασμός μεθόδων Monte Carlo και δυναμικού προγραμματισμού (Sutton και Barto, 1998). Είναι η πιο διαδεδομένη μέθοδος ενισχυτικής μάθησης και ασχολείται με το πρόβλημα της καθυστερημένης ανταμοιβής, δηλαδή το πώς θα κατανέμει την αμοιβή σε ενέργειες που παρήγαγαν την τελική ανταμοιβή. Αυτό το είδος των μεθόδων

επιτυγχάνει μάθηση μετ' εμπειρίας χωρίς την γνώση του μοντέλου εκ των προτέρων. Ενημερώνει τη συνάρτηση αξίας κατά την εμπειρία και βασίζεται σε προηγούμενες εμπειρίες για την νέα εκτίμηση-εξίσωση 3.1.

$$V(S_t) = V(S_t) + \alpha \cdot [r_{t+1} + \gamma \cdot V(S_{t+1}) - V(S_t)] \quad (3.1)$$

Όπου $V(S_t)$ η συνάρτηση κατάστασης-αξίας για κατάσταση S τη χρονική στιγμή t, α ο ρυθμός μάθησης, r_{t+1} η αμοιβή τη χρονική στιγμή t+1, γ ο ρυθμός έκπτωσης και $V(S_{t+1})$ η συνάρτηση κατάστασης-αξίας για κατάσταση S τη χρονική στιγμή t+1. Άρα η αξία της τρέχουσας κατάστασης ενημερώνεται με βάση την επόμενη κατάσταση (bootstrapping).

Σε αυτές τις μεθόδους υπάρχει και ένα σφάλμα δt (εξίσωση 3.2), όπου rt+1 το ενισχυτικό σήμα της χρονικής στιγμής t+1, γ ο ρυθμός έκπτωσης, V(St) η συνάρτηση κατάστασης-αξίας για κατάσταση S τη χρονική στιγμή t και V(St+1) η συνάρτηση κατάστασης-αξίας για κατάσταση S τη χρονική στιγμή t+1.

$$\delta_t = r_{t+1} + \gamma \cdot V(S_{t+1}) - V(S_t) \quad (3.2)$$

Επίσης υπάρχει το λεγόμενο «ίχνος επιλεξιμότητας» το οποίο κατανέμει την χρονική ευθύνη σε καταστάσεις οι οποίες είναι συχνά επισκέψιμες, ενώ σε προηγούμενες καταστάσεις ή σε καταστάσεις που παύουν από ένα σημείο και μετά να είναι επισκέψιμες εξαφανίζεται σταδιακά. Η εξίσωση (3.3) παρουσιάζει αυτόν το μηχανισμό.

$$e_t(s) = \begin{cases} \gamma\lambda e_{t-1}(s), & s \neq s_t \\ \gamma\lambda e_{t-1}(s) + 1, & s = s_t \end{cases} \quad (3.3)$$

Όπου $e_t(s)$ ίχνος επιλεξιμότητας της κατάστασης s τη χρονική στιγμή t, γ ο ρυθμός έκπτωσης και λ η παράμετρος εξαφάνισης ίχνους.

Τέλος με την εξίσωση 3.4 παρουσιάζεται η αλλαγή της συνάρτησης κατάστασης-αξίας για τις πρόσφατα επισκεφθείσες καταστάσεις. Όπου α ο ρυθμός μάθησης, δ_t το σφάλμα χρονικών διαφορών (εξίσωση 3.2) και $e_t(s)$ ίχνος επιλεξιμότητας (εξίσωση 3.3).

$$\Delta V(s) = \alpha\, \delta_t\, e_t(s) \quad (3.4)$$

3.3.3.1 Q-Learning

Προτάθηκε από τον Watkins το 1989. Είναι μια βασική και γνωστή μέθοδος ΕΜ χωρίς απαραίτητα την χρήση μοντέλου (model-free). Χρησιμοποιείται για να υπολογιστεί η βέλτιστη συνάρτηση Q για κάθε ενέργεια και για κάθε κατάσταση. Χρησιμοποιεί μέθοδο ΧΔ εκτός-πολιτικής και ουσιαστικά υπολογίζει την τιμή Q(s,a), δηλαδή, την αξία της κάθε μετάβασης κατάστασης-ενέργειας. Συγκλίνει αποδεδειγμένα στη βέλτιστη πολιτική (υπό κάποιες προϋποθέσεις) ασχέτως με την πολιτική συμπεριφοράς (behaviour policy) που χρησιμοποιείται.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\right] \quad (3.2)$$

3.3.3.2 SARSA (State Action Reward State Action)

Προτάθηκε από τους Rummery & Niranjan το 1994. Είναι μια μέθοδος ΕΜ, όπου χρησιμοποιείται για να βρει την βέλτιστη συνάρτηση Q για την τρέχουσα πολιτική, για κάθε επιλεγμένη ενέργεια και για κάθε κατάσταση. Αποτελεί μέθοδο ΧΔ εντός-πολιτικής και ουσιαστικά βρίσκει την τιμή Q(s,a), δηλαδή, την αξία της κάθε μετάβασης κατάστασης-ενέργειας (όπως ο Q-Learning (Watkins, 1989)) αλλά ταυτόχρονα αλλάζει και την πολιτική. Για να υπολογιστεί η αξία μιας ενέργειας (Q(s,a)) την παρούσα χρονική στιγμή πρέπει να ληφθεί υπόψη η τρέχουσα κατάσταση s, η επιλεγμένη ενέργεια α, η αμοιβή r που λήφθηκε, η επόμενη κατάσταση s'και η επόμενη επιλεγμένη ενέργεια α΄.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\right] \quad (3.3)$$

3.3.3.3 Policy Hill-Climbing (PHC)

Προτάθηκε από τους Bowling & Veloso το 2002. Αποτελεί επέκταση του Q-Learning (Watkins, 1989) και συνήθως εφαρμόζεται σε παιχνίδια μικτής στρατηγικής. Σε κάθε βήμα του αλγόριθμου PHC, επιλέγεται μια ενέργεια με βάση την κατανομή μιας πιθανότητας. Στη συνέχεια ανανεώνεται η αξία της επιλεγμένης ενέργειας με βάση την αμοιβή που λήφθηκε και την επόμενη κατάσταση. Η πιθανότητα της επιλεγμένης ενέργειας επίσης αλλάζει με αύξηση

της κατά ένα ρυθμό μάθησης δ, όπου δ ϵ (0,1]. Ο αλγόριθμος συγκλίνει σε μια βέλτιστη πολιτική.

3.3.3.4 WoLF Policy Hill-Climbing (WoLF- PHC)

Προτάθηκε από τους Bowling & Veloso το 2002. Είναι επέκταση του προηγούμενου αλγόριθμου. Πιο συγκεκριμένα, ο αλγόριθμος λειτουργεί με τον εξής τρόπο: όταν ο πράκτορας κερδίζει κάνει μικρά βήματα (δ είναι μικρό) και η πολιτική παραμένει ίδια, ενώ όταν ο πράκτορας χάνει κάνει μεγαλύτερα βήματα (δ αυξάνεται) και αλλάζει την πολιτική του για να την καλυτερεύσει. Νίκη για τον πράκτορα σημαίνει ότι πήρε αμοιβή μεγαλύτερη απ' την μέση αμοιβή που έχει λάβει μέχρι στιγμής. Αυτή η μεθοδολογία βοηθά τον παίχτη να προσαρμοστεί γρήγορα στις αλλαγές. Σε αυτό τον αλγόριθμο γίνεται χρήση μεταβλητού ρυθμού μάθησης σε αντίθεση με τον PHC (Bowling & Veloso 2002).

Κεφάλαιο 4

Διάσπαση HEXQ

Σ’ αυτό το κεφάλαιο στόχος είναι η θεωρητική περιγραφή του αλγόριθμου HexQ αλλά και η παραπομπή στις σχετικές συναρτήσεις, δομές που χρησιμοποιήθηκαν κατά την υλοποίηση (σημειώνονται σε παρενθέσεις σε ιταλική μορφή). Ο αλγόριθμος αυτό χωρίζει το state-space ενός πολυδιάστατου, πεπερασμένου, Μαρκοβιανού προβλήματος, σε μια ιεραρχία από υπο-προβλήματα βάσει κάποιων συγκεκριμένων κανόνων διάσπασης. Αυτοί οι ξεχωριστοί κανόνες διάσπασης που απαρτίζουν τον αλγόριθμο HexQ, οδηγούν στην αυτοματοποίηση της διάσπασης του προβλήματος, σε αντίθεση με άλλους ιεραρχικούς αλγόριθμους όπου ο προγραμματιστής καλείται ο ίδιος να ορίσει την ιεραρχία του προβλήματος. Ο αλγόριθμος αυτός ονομάζεται HexQ λόγω της μεθοδολογίας και των συνθηκών διάσπασης που ακολουθεί για να λύσει ένα οποιοδήποτε πολυδιάστατο πρόβλημα MDP.

Μέσω του HexQ το πολυδιάστατο MDP διασπάται σε ένα δέντρο που αποτελείται από μικρότερα MDPs (sub-MDPs). Δημιουργεί μια ιεραρχία επιπέδων όπου στο κάθε επίπεδο εφαρμόζεται μια πολιτική. Ο αριθμός των επιπέδων εξαρτάται από τον αριθμό των μεταβλητών του προβλήματος (πρόκειται να επεξηγηθεί περαιτέρω στο σχετικό υποκεφάλαιο). Η πολιτική του κάθε επιπέδου, είναι

συγχρόνως η αφαιρετική ενέργεια του επόμενου (υψηλότερου επιπέδου). Άρα η πολιτική του προηγούμενου επιπέδου, είναι μια σειρά κινήσεων (μια αφαιρετική ενέργεια) για το επόμενο επίπεδο. Στο κατώτατο επίπεδο δεν υπάρχουν αφαιρετικές ενέργειες παρά μόνο primitive ενέργειες, μοναδικές κινήσεις που επιστρέφουν μια αμοιβή ανά ενέργεια. Όταν ο έλεγχος πλέον επιστραφεί στο ανώτατο επίπεδο, το MDP έχει λυθεί.

4.1 ΥΠΟΘΕΣΕΙΣ

Το πρόβλημα του ταξί (TP), στο οποίο εφαρμόστηκε ο αλγόριθμος HexQ, είναι ένα επεισοδιακό και μη-στοχαστικό πρόβλημα (ο HexQ λύνει σαφώς και στοχαστικά, πεπερασμένα προβλήματα). Οι αμοιβές του προβλήματος έχουν οριστεί όλες με αρνητικές τιμές, παρά μόνο η αμοιβή που θα δίνεται κατά τον επιτυχή τερματισμό του προβλήματος έχει θετική τιμή. Ο ρυθμός έκπτωσης έχει οριστεί ίσος με ένα (ο εκπαιδευόμενος θεωρείται πλήρως διορατικός στα πλαίσια της εκπαίδευσης του). Στόχος στο πρόβλημα αυτό είναι η λύση του, σε ελάχιστο χρόνο. Οπόταν μέσω μιας πολιτικής που ακολουθείται σε κάθε επίπεδο του δέντρου, το MDP λύνεται με πιθανότητα 1 ξεκινώντας από οποιοδήποτε αρχική κατάσταση του προβλήματος.

4.2 ΙΕΡΑΡΧΙΑ HEXQ

Σ’ αυτό το υποκεφάλαιο θα επεξηγηθεί πώς χωρίζοντας το σύνολο καταστάσεων του προβλήματος, το MDP διαχωρίζεται σε περιοχές και πώς οι περιοχές αυτές που απαρτίζονται από μικρότερους υπό-χώρους (sub-MDPs) οδηγούν στην λύση του προβλήματος.

Για να λυθεί το πρόβλημα αποδοτικά, γίνεται χρήση και αποθήκευση πολιτικών της κάθε περιοχής. Πολιτική αποτελεί μία σειρά από ενέργειες που σταδιακά οδηγούν με πιθανότητα ένα, στον τερματισμό του προβλήματος, στην λύση του MDP, ξεκινώντας από οποιαδήποτε αρχική κατάσταση. Άρα οι πολιτικές που χρησιμοποιούνται σε ένα επίπεδο και συγκεκριμένα σε ένα sub-MDP, αποτελούν αφαιρετικές ενέργειες και αποθηκεύονται σαν αφαιρετικές ενέργειες, για να χρησιμοποιηθούν αναλόγως στο υψηλότερο επίπεδο της ιεραρχίας. Λύνοντας το sub-MDP του επιπέδου που βρίσκεται ο εκπαιδευόμενος, ο έλεγχος, η ροή του προβλήματος μεταφέρεται στο ανώτερο επίπεδο και ιεραρχικά λύνεται το πρόβλημα. Κατά αυτόν τον τρόπο η μάθηση μεταδίδεται από το ένα επίπεδο στο άλλο, μέσω της λύσης των υπο-προβλημάτων.

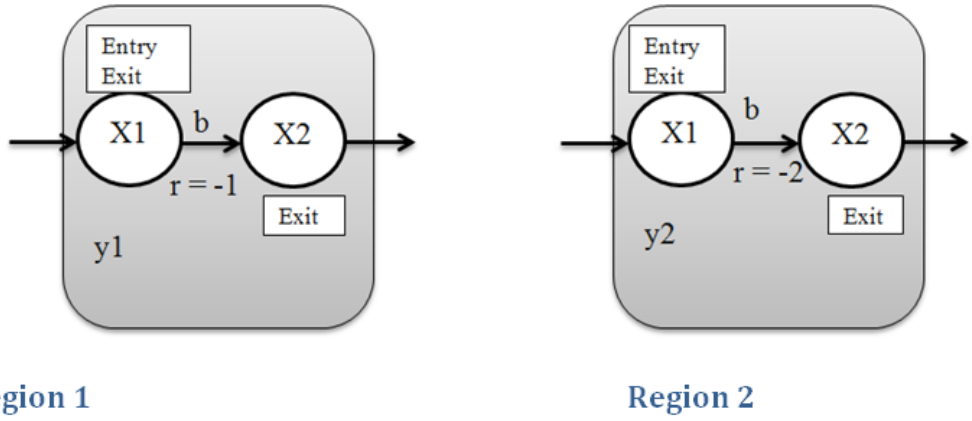
4.2.1 Διαχωρισμός του προβλήματος

Το TP αποτελεί ένα MDP πρόβλημα, *τριών* διαστάσεων (3-dimensional MDP). Συνεπώς ο χώρος καταστάσεων θα αποτελείται από τρεις μεταβλητές (οι οποίες αποθηκεύονται στο *ArrayList X* μέτά από κλήση της *initializeX()(δες ΠΑΡΑΡΤΗΜΑ Α, σελ. 25)*) και θα είναι της μορφής: (x,y,z). Για να γίνει ανάλογο του προβλήματος του ταξί, οι μεταβλητές του χώρου καταστάσεων ονομάζονται: (taxi location, passenger location, destination location). Άρα η ιεραρχία θα αποτελείται από τρία επίπεδα (ένα επίπεδο ανά μεταβλητή και *depth=X.size*). Στόχος του αλγόριθμου είναι ανά μεταβλητή (μία εκ των τριών κάθε φορά), να διαχωρίσει σε περιοχές το σύνολο καταστάσεων. Έτσι επιτρέπει μέσω μεταβάσεων, την επικοινωνία των περιοχών του ίδιου επιπέδου.

4.2.2 Έξοδοι

Στον αλγόριθμο HexQ υπάρχει η έννοια του συνόλου των εξόδων (είναι αποθηκευμένες στο *ArrayList* Exits και βρέθηκαν μετά από κλήση της συνάρτησης *findExits(e) (δες ΠΑΡΑΡΤΗΜΑ Α, σελ. 8)).* Έξοδος αποτελεί ένα ζεύγος τύπου (κατάσταση s, ενέργεια a), στο οποίο ο οποίο ο πράκτορας έχει πρόσβαση από οποιαδήποτε αρχική κατάσταση. Βρισκόμενος στην κατάσταση s, επιλέγοντας την ενέργεια a, προκαλείται μια διαφορετική, απρόβλεπτη μετάβαση (που δεν είναι γνωστή από την συνάρτηση μεταβάσεων). Πώς ξεχωρίζουμε τότε απρόβλεπτη μετάβαση, από μία κανονική μετάβαση; Ένα ζεύγος (s,a) είναι έξοδος όταν η μετάβαση αυτή οδηγεί σε:

- αλλαγή της περιοχής που βρίσκεται ο πράκτορας.
- αλλαγή της τιμής κάποιας από τις υπόλοιπες μεταβλητές του προβλήματος (όπως προαναφέρθηκε, ο αλγόριθμος χειρίζεται μια μεταβλητή κάθε φορά σε σχέση με μια άλλη μεταβλητή όπου διατηρείται σταθερή η τιμή της). ✕
- Διαφορετική συνάρτηση αμοιβής σε ίδιες τιμές της μεταβλητής x αλλά σε διαφορετικές τιμές της μεταβλητής y (σχήμα 4.1).
- Διαφορετική συνάρτηση μεταβάσεων σε ίδιες τιμές της μεταβλητής x αλλά σε διαφορετικές τιμές της μεταβλητής y.
- Τερματισμό του προβλήματος (λύση του ολικού MDP). ✕

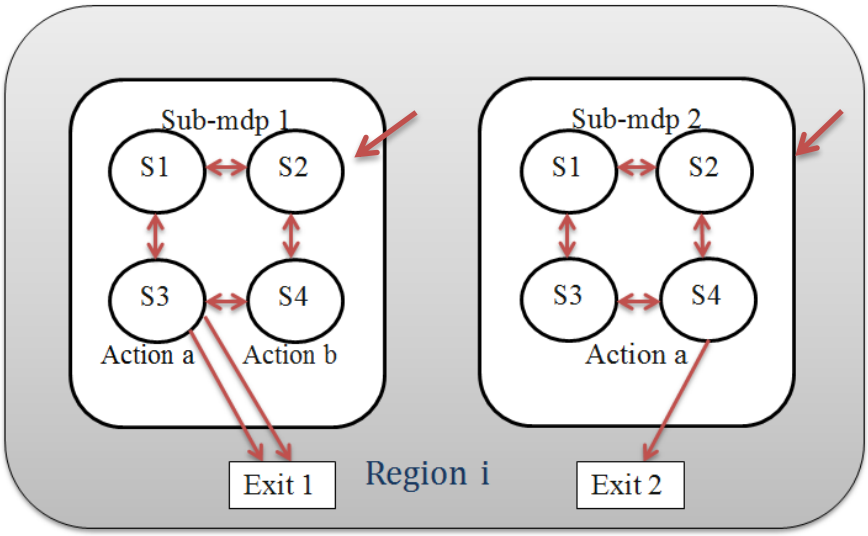


Σχήμα 4.1: Στο σχήμα απεικονίζεται η περίπτωση όπου υπάρχει διαφορετική συνάρτηση αμοιβής σε ίδιες τιμές της μεταβλητής x αλλά σε διαφορετικές τιμές της μεταβλητής y. Όπως παρατηρούμε το ((x1,y1), b) δίνει αμοιβή r=-1 ενώ το ((x1,y2), b) δίνει αμοιβή r=-2. Αυτός είναι ένας από τους ορισμούς της εξόδου, άρα το ((x1,y1), b) και ((x1,y2), b) αποτελούν ζεύγη εξόδων.

Οι περιπτώσεις που συναντήθηκαν στο TP έχουν ✕ στο τέλος.

4.2.3 Είσοδοι

Στον αλγόριθμο HexQ εισάγεται και η έννοια των εισόδων (αποθηκεύτηκαν στο *ArrayList Entries* και βρέθηκαν μέσω της *συνάρτησης findEntries(e) (δες ΠΑΡΑΡΤΗΜΑ Α, σελ. 10)).* Το σύνολο το εισόδων που εμπλέκεται στην διαδικασία του αλγορίθμου, αποτελεί το σύνολο των αρχικών καταστάσεων του MDP (οι καταστάσεις που μπορεί αρχικά να βρεθεί ο εκπαιδευόμενος). Επίσης κατάσταση που αποτελεί είσοδο, είναι η κατάσταση που ακολουθεί την επιλογή ζεύγους εξόδου, δηλαδή η αμέσως επόμενη κατάσταση μετά από την εκτέλεση μίας εξόδου. Για παράδειγμα, αν το (s,a) αποτελεί έξοδο τότε το s' είναι είσοδος, αν ισχύει (s,a)->s' όπου $T^{a}_{ss'} > 0$.

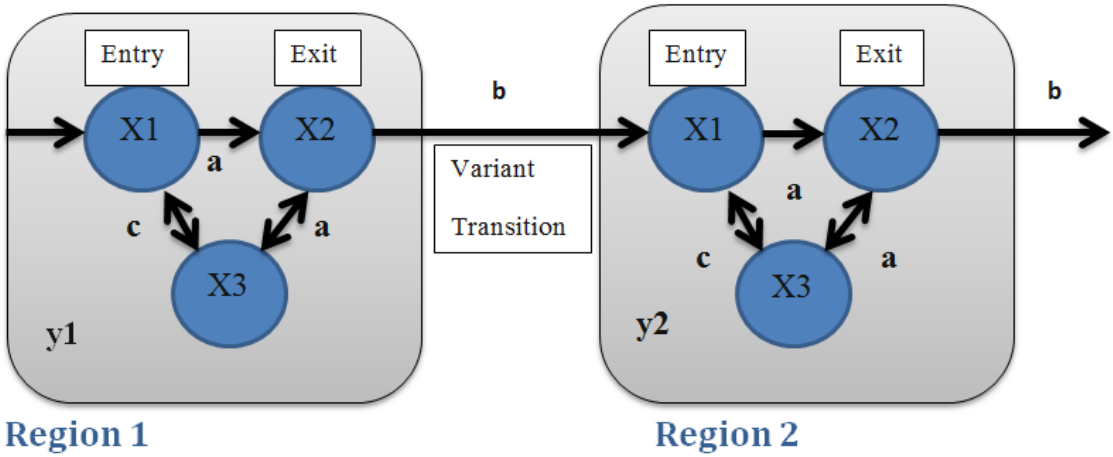


Σχήμα 4.2:Το πιο πάνω σχήμα απεικονίζει μια περιοχή που αποτελείται από δυο sub-MDPs. Το κάθε sub-MDP δημιουργείται στην παρουσία κατάστασης εξόδου και μόνο (στην περίπτωση που μια κατάσταση οδηγεί σε έξοδο με περισσότερες από μια ενέργειες (όπως στο sub-MDP 1), τότε ένα sub-MDP δημιουργείται μόνο, δηλαδή ο αριθμός των sub-MDPs δεν εξαρτάται από τον αριθμό των ενεργειών μίας κατάσταση που οδηγεί σε έξοδο.

4.2.4 Περιοχές

Πιο πάνω αναφέραμε ότι μια βασική διαδικασία του HexQ, που έχει μεγάλη σημασία όσο αφορά τη λύση του MDP, είναι ο διαχωρισμός του MDP σε περιοχές (βρέθηκαν μέσω της *συνάρτησης Regions(e) (δες ΠΑΡΑΡΤΗΜΑ Α, σελ. 13)* και αποθηκεύτηκαν στο *ArrayList MERs,* βρέθηκε ένα σε κάθε επίπεδο). Τι εννοούμε με τον όρο περιοχές; Ο HexQ χωρίζει της καταστάσεις του MDP και αυτός ο διαχωρισμός οδηγεί στην δημιουργία μπλοκ, τις λεγόμενες περιοχές όπως φαίνεται χαρακτηριστικά στο σχήμα 4.3. Σε κάθε περιοχή, ο εκπαιδευόμενος μπορεί να μεταβεί σταδιακά (με μια πεπερασμένη σειρά βημάτων) από οποιαδήποτε κατάσταση εισόδου και με σειρά από μεταβάσεις που δεν προκαλούν έξοδο (non-exit transitions), σε κατάσταση εξόδου, με πιθανότητα ένα.

Εάν δεν ισχύει αυτός ο κανόνας και υπάρξει μετάβαση εξόδου τότε έχουμε interregional transition. Στο TP βρέθηκε ένα MER και στα δυο επίπεδα.



Σχήμα 4.3: Στο σχήμα απεικονίζονται με μπλε κύκλους οι καταστάσεις της κάθε περιοχής, τα μαύρα βέλη δείχνουν τις μεταβάσεις μεταξύ των καταστάσεων και τα γκρίζα στρογγυλεμένα ορθογώνια απεικονίζουν τις περιοχές. Τα y1 και y2 απεικονίζουν το σύμβολο της μεταβλητής πλαισίου (context variable) και τα a, b και c είναι ενέργειες. Όπως παρατηρούμε, δυο περιοχές επικοινωνούν μεταξύ τους μέσω μιας διαφορετικής μετάβασης, μιας μετάβασης εξόδου, διότι στο region 1 το ζεύγος (x2, b) αποτελεί έξοδο. Για παράδειγμα παρατηρούμε επίσης ότι με την επιλογή της ενέργειας b στην κατάσταση x2 του region 1, αλλάζει η τιμή της μεταβλητής y από y=y1 σε y=y2. Αυτό όπως προαναφέρθηκε είναι ένας από τους ορισμούς της εξόδου. Επίσης παρατηρούμε ότι, η κατάσταση x1 του region 2, αποτελεί είσοδο, διότι με βάση τον ορισμό της εισόδου, είναι κατάσταση η οποία προήλθε από έξοδο. Η κατάσταση x1 του region 1, αποτελεί είσοδο, διότι προφανώς είναι μία από τις αρχικές καταστάσεις του προβλήματος. Σε κάθε επίπεδο, ο εκπαιδευόμενος κινείται στις καταστάσεις που αντιστοιχούν στην τρέχουσα μεταβλητή (τρέχον επίπεδο). Οπόταν αν ο εκπαιδευόμενος προκαλέσει μια έξοδο στο επίπεδο, συγχρόνως ο εκπαιδευόμενος αλλάζει επίπεδο λόγω της πρόκλησης εξόδου.

4.2.5 Sub-MDPs

Τα sub-MDPs δημιουργούνται από τον ίδιο τον αλγόριθμο (αποθηκέυτηκαν στο *ArrayList Sub_Mdps και βρέθηκαν μέσω της συνάρτησης constructSubMdps(e) (δες ΠΑΡΑΡΤΗΜΑ Α, σελ. 27)* και *constructTopLevelSub(d)) (δες ΠΑΡΑΡΤΗΜΑ Α, σελ. 30),* αυτόματα και όταν λυθούν, στο τελευταίο στάδιο του αλγορίθμου, λύνεται το ολικό MDP, συνεπώς και το πρόβλημα. Για την δημιουργία αυτών των υπο-χώρων χρειάζεται: το σύνολο καταστάσεων της περιοχής, η συνάρτηση μεταβάσεων και η συνάρτηση αμοιβής του MDP. Οπόταν σε κάθε περιοχή ενδέχεται να υπάρχουν πολλαπλά sub-MDPs. Το ερώτημα που προκύπτει είναι, πόσα sub-MDPs δημιουργεί αλγόριθμος σε κάθε περιοχή; Όπως φαίνεται και στο σχήμα (4.2), δημιουργούνται τόσα sub-MDPs όσα και ο αριθμός των καταστάσεων που οδηγούν σε έξοδο του sub-MDP κάθε περιοχής. Η έξοδος αυτή οδηγείται με μηδενική αμοιβή σε ένα absorbing state (το οποίο είναι ένα νοητό state και δείχνει ότι επήλθε τερματισμός του sub-MDP). Ο λόγος που ονομάστηκαν sub-MDPs είναι καθαρά για λόγους διαφοροποίησης τους από το ολικό MDP. Για να λυθεί ένα sub-MDP και να αφήσει ο εκπαιδευόμενος την τρέχουσα περιοχή, πρέπει αναγκαστικά (κανόνας sub-MDP) να περάσει από την έξοδο και να εκτελέσει την ενέργεια που θα τον μεταφέρει σε άλλη περιοχή ή στο ανώτερο επίπεδο. Όσο αφορά το τελευταίο (ανώτατο) επίπεδο, υπάρχει ένα και μοναδικό sub-MDP, το οποίο όταν λυθεί, λύνεται και το ολικό MDP. Αποτελείται από καταστάσεις του ολικού MDP και από αφαιρετικές ενέργειες που αντικατοπτρίζουν πολιτικές κατώτερων sub-MDPs. Ουσιαστικά το ανώτατο sub-MDP καλεί εξ' ορισμού τις πολιτικές κατώτερων sub-MDPs (τα abstract actions δηλαδή) μέχρι να βρει έξοδο και να επιλέξει άλλο sub-MDP. Όταν η ροή επανέλθει στο ανώτατο επίπεδο τότε το πρόβλημα έχει λυθεί. Άρα η συνάρτηση κατάστασης-αξίας αυτού που περιγράφηκε πιο πάνω αντικατοπτρίζεται απ' την εξίσωση (4.1).

$$V_m^{\pi}(s) = E\{\sum_{n=1}^{N-1} r_n + \sum_{n=N}^{\infty} r_n\}$$

(4.1)

Η συνάρτηση κατάστασης-αξίας (4.1) δείχνει το αναμενόμενο άθροισμα N μελλοντικών primitive ενεργειών με βάση μια πολιτική π, όταν ο πράκτορας, αρχίζοντας από μια κατάσταση s, εκτελεί ένα abstract action και τερματίζει βρισκόμενος σε μια κατάσταση s'. Το 1^{ο} μέρος της εξίσωσης δείχνει το άθροισμα των αμοιβών μέχρι την έξοδο (εκτέλεση μιας αφαιρετικής ενέργειας εμπλέκει περισσότερες από μία primitive ενέργειες, έστω N) και το 2^{ο} μέρος δείχνει την αξία της κατάστασης s' που ακολουθεί την έξοδο.

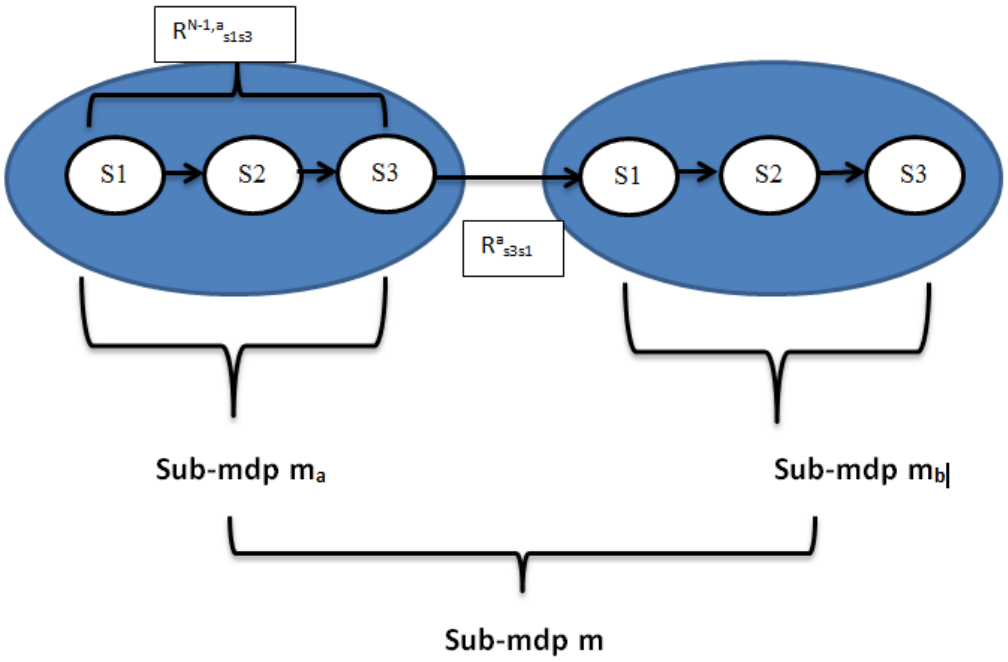
$$V_m^{\pi}(s) = \sum_{s',N} T_{ss'}^{Na}[R_{ss'}^{Na} + V_m^{\pi}(s')]$$

(4.2)

Η εξίσωση (4.2) δείχνει επίσης την συνάρτηση κατάστασης-αξίας του ανώτατου sub-MDP, το ίδιο ακριβώς που φαίνεται και στην εξίσωση (4.1).

4.2.6 Ιεραρχικές Πολιτικές

Ιεραρχική πολιτική είναι ένα σύνολο από πολιτικές, μια πολιτική για κάθε sub-MDP, στο δέντρο του HexQ (το *ArrayList Sub_Mdps* περιέχει τα sub-MDPs όλων των επιπέδων άρα θεωρείται το δέντρο ιεραρχίας). Άρα έστω S το σύνολο των sub-MDPs, τότε θα έχουμε: S={s_1, s_2, …s_n} και σύνολο Π={π_1, π_2, …π_n} με n πολιτικές.



Σχήμα 4.4: Στο σχήμα είναι ένα παράδειγμα όπου το sub-MDP m ακολουθεί μια πολιτική π_m. Το sub-MDP m καλεί το sub-MDP m_a με abstract action a, εκτελούνται τρία βήματα και παρατηρούμε το άθροισμα των αμοιβών μέχρι και την έξοδο απ' το συγκεκριμένο sub-MDP. Το $R^{2,a}_{s1s3}$ είναι για τα δυο πρώτα βήματα και το R^{a}_{s3s1} για το τελευταίο βήμα. Η συνάρτηση R που δίνει τις αμοιβές είναι *η takeAction(int action, State_Variables s.)*

Με βάση το σχήμα (4.4) πηγάζει και η πιο κάτω εξίσωση:

$$V_m^{\pi}(s) = V_{m_a}^{\pi}(s) + \sum_{s'} T^{a}_{s_a s'}[R^{a}_{s_a s'} + V_m^{\pi}(s')] \tag{4.3}$$

4.2.7 Συνάρτηση E

Σε μια κατάσταση s για abstract action a, σ' ένα sub-MDP m, είναι η αναμενόμενη τιμή των μελλοντικών αμοιβών μετά την εκτέλεση του a, από το s με βάση μια πολιτική π (Το κάθε sub-MDP περιέχει στη δομή του ένα δυναμικό δισδιάστατο πίνακα E [καταστάσεις^e][ενέγειες^e] που αντιστοιχεί στη συνάρτηση E). Περιέχει το αναμενόμενο primitive reward της εξόδου, αλλά δεν περιέχει αμοιβές που συναθροίστηκαν κατά την εκτέλεση μιας αφαιρετικής ενέργειας a στο sub-MDP m.

$$E_m^\pi(s,a) = \sum_{s'} T^a_{s_a s'}[R^a_{s_a s'} + V_m^\pi(s')]$$

(4.4)

Αντικαθιστώντας την εξίσωση (4.4) με την εξίσωση (4.3) προκύπτει η πιο κάτω εξίσωση:

$$V_m^\pi(s) = V_{m_a}^\pi(s) + E_m^\pi(s,a).$$

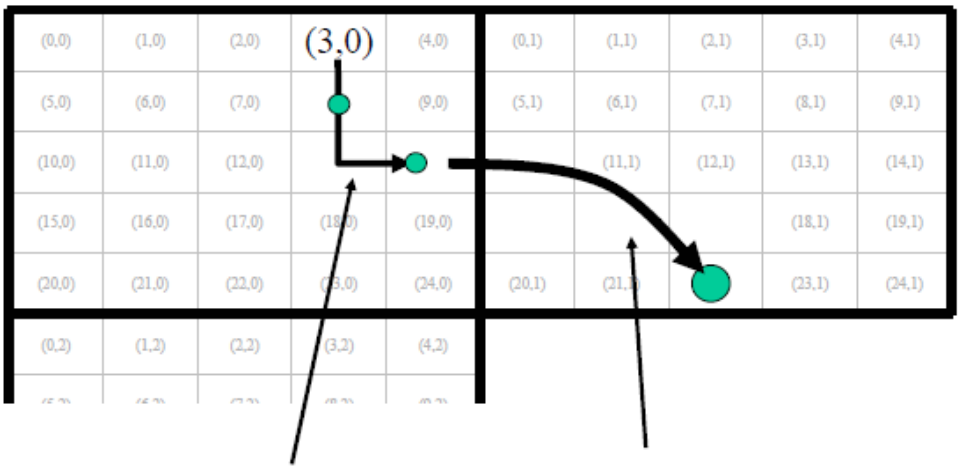
(4.5)

Σε μονοδιάστατο MDP ή στο 1^o επίπεδο της ιεραρχίας, η συνάρτηση E είναι πανομοιότυπη με την συνάρτηση Q του $V_m^\pi(s) = E_m^\pi(s,a) = Q_m^\pi(s,a)$ αλγόριθμου MAXQ που προτάθηκε από τον Dietterich το 2000. Σ' αυτές τις περιπτώσεις το 1^o μέρος του αθροίσματος της εξίσωσης (4.5) είναι ίσο με μηδέν. Άρα όπου α=π(s). Η συνάρτηση E είναι μια ιεραρχική γενίκευση της συνάρτησης Q. Οπόταν εφ' όσον τονίστηκε στα πιο πάνω υποκεφάλαια ο αναδρομικός χαρακτήρας του αλγορίθμου HexQ, έχουμε ότι το δέντρο που δημιουργήθηκε

(βάθους d-1), καλεί σειρά από abstract actions a_d, a_{d-1}, …a_1 με τα αντίστοιχα sub-MDPs m_d, m_{d-1}, … m_1 και προκύπτει η εξής συνάρτηση αξίας:

$$V_d^{\pi}(s) = E_1^{\pi}(s, a_1) + E_2^{\pi}(s, a_2) + \ldots + E_d^{\pi}(s, a_d). \tag{4.6}$$

Μια εφαρμογή της εξίσωσης (4.6) δείχνει το σχήμα (4.5):



Σχήμα 4.5: Στο σχήμα παρατηρούμε την συνάρτηση αξίας μιας εφαρμογής ενός άλλου παιχνιδιού. Ο πράκτορας βρίσκεται στην κατάσταση (3,0), είναι ένα δισδιάστατο MDP, όπου το σύνολο καταστάσεων αναπαριστάται ως εξής: (room position, room number). Οπόταν βλέπουμε την συνάρτηση αξίας από την κατάσταση (3,0) μέχρι και την έξοδο (στη συγκεκριμένη περίπτωση και τερματισμό συγχρόνως). Η αφαιρετική ενέργεια που επιλέχθηκε είναι αυτό που θα οδηγήσει τον πράκτορα έξω από την ανατολική έξοδο. Κάθε βήμα έχει αμοιβή ίση με -1 (οπόταν για να βγει από την έξοδο παίρνει αμοιβή ίση με -3 και μέχρι να φτάσει στο στόχο του παίρνει αμοιβή ίση με -5). Η εικόνα αυτή έχει παρθεί απ' τον Hengst (2002).

4.2.8 Πολυδιάστατα MDP (> 2-d)

Ο αλγόριθμος σε περίπτωση που το πρόβλημα εμπλέκει περισσότερες από δυο μεταβλητές, αποκτά αναδρομικό χαρακτήρα. Έστω ότι υπάρχουν d μεταβλητές, τότε έχουμε χωρίζουμε δυο σύνολα ως εξής: $S=\{s^1, s^2, \ldots, s^{d-1}\}$ και $S'=\{s^d\}$. Τότε ορίζεται ένα νέο σύνολο καταστάσεων που παράγεται από το καρτεσιανό γινόμενο των συστατικών του S, με σταθερό το συστατικό του S', δηλαδή $s=(s^1x s^2x \ldots x s^{d-1} , s^d)$. Στη συνέχεια το $s=(s^1x s^2x \ldots x s^{d-2}, s^{d-1})$ κοκ μέχρι να γίνει διάσπαση του προβλήματος αναδρομικά και να τερματίσει όταν καταλήξει σε μια μεταβλητή κατάστασης. Για κάθε διάσπαση sub-MDP απαλείφεται και μια μεταβλητή. Η διάσπαση του προβλήματος σε πολυδιάστατα MDP είναι η πιο πάνω, έτσι δημιουργείται ιεραρχικά το δέντρο βάθους d-1, που αντιστοιχεί στο ολικό sub-MDP. Οι διαδικασίες και οι κανόνες που αναγράφονται σ' αυτό το κεφάλαιο ισχύουν ως έχουν και σε αυτού του τύπου τα MDPs.

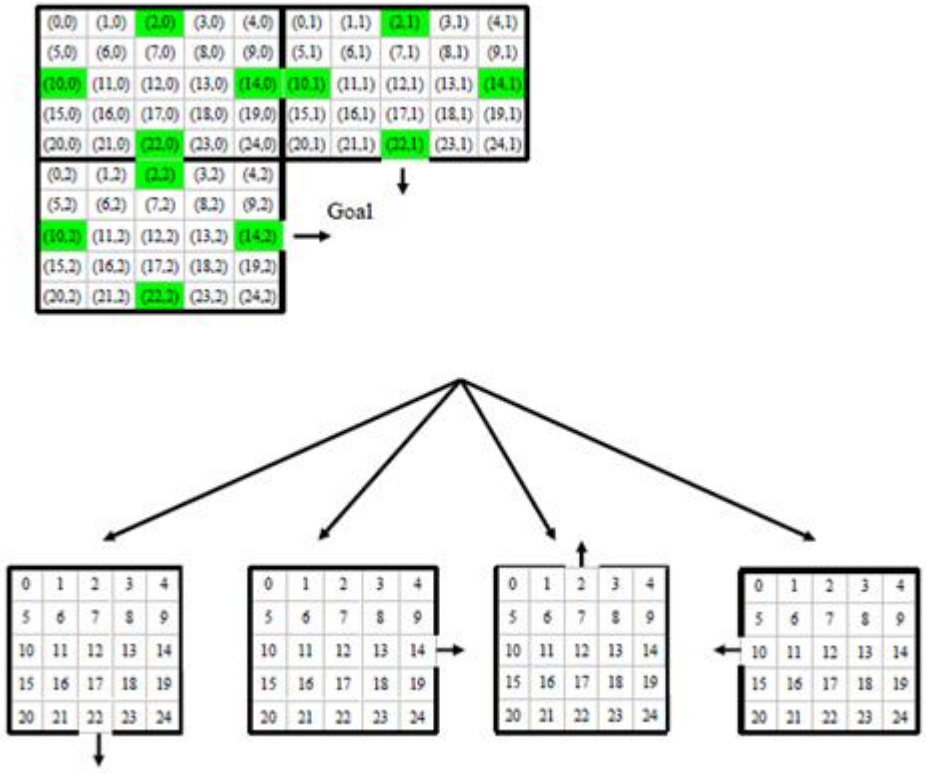
4.3 ΣΥΜΠΑΓΗΣ ΑΝΑΠΑΡΑΣΤΑΣΗ ΤΩΝ ΔΟΜΩΝ ΤΟΥ HEXQ

Αναμφισβήτητα ο αλγόριθμος HexQ, όπως θα δούμε και σε περαιτέρω κεφάλαια, υπερέχει λόγω εξοικονόμησης υπολογιστικού χρόνου αλλά και μνήμης. Σ' αυτό το υποκεφάλαιο θα δείξουμε πώς το state abstraction δίνει πλεόνασμα στον αλγόριθμο.

4.3.1 Μαρκοβιανές Ισοδύναμες Περιοχές (Markov Equivalent Regions)

Η έννοια της περιοχής επεξηγήθηκε πιο πάνω και είναι έννοια ισοδύναμη του MER. Έστω ότι έχουμε ένα δισδιάστατο MDP, με σύνολο μεταβλητών S=(x, y). Τότε

παρατηρούμε ότι υπάρχουν πολλές ίδιες περιοχές (Markov Equivalent Regions, MERs). Βλέποντας το σχήμα (4.6) με βάση όλα αυτά τα οποία αναφέρθηκαν σε αυτό το κεφάλαιο, θα υποθέταμε ότι θα δημιουργηθούν 12 sub-MDPs, 4 για κάθε ορθογώνιο δωμάτιο που συμβολίζει μια περιοχή, λόγω της παρουσίας τεσσάρων εξόδων στο δωμάτιο. Όμως ο αλγόριθμος HexQ λόγω του ότι τα sub-MDPs θα ήταν καθαρά διπλότυπα λόγω της κοινής τιμής της μεταβλητής x (αφού μόνο στη y μεταβλητή διαφέρουν), απαλείφει τα οκτώ απ' τα δώδεκα sub-MDPs. Ως αποτέλεσμα δεν υπάρχουν διπλότυπες δομές, άρα υπερέχει σε μνήμη αλλά και σε υπολογιστικό χρόνο.



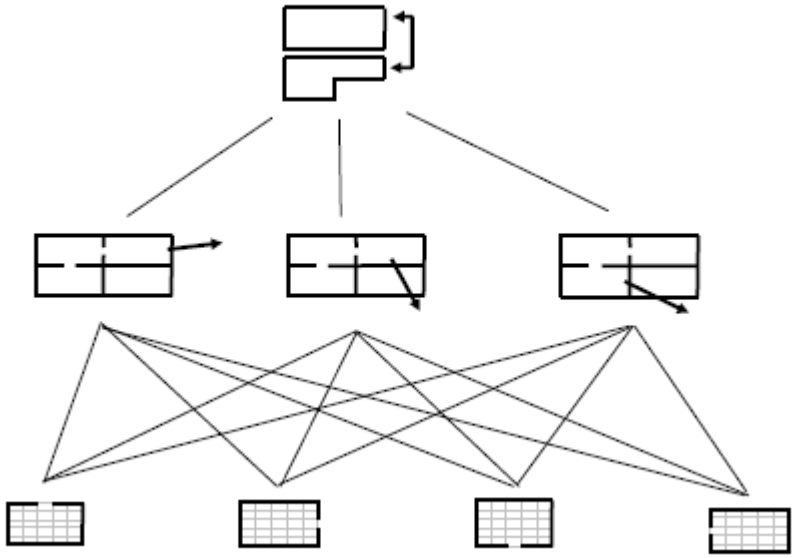
Σχήμα 4.6: Ο αλγόριθμος αυτός απαλείφει τα διπλότυπα sub-MDPs με αποτέλεσμα να μετατρέψει τον HexQ γράφο σε ένα συμπαγή, άκυκλο, κατευθυνόμενο γράφο. Δημιουργούνται 12 sub-MDPs, 4 για κάθε ορθογώνιο δωμάτιο που συμβολίζει μια περιοχή, λόγω της παρουσίας τεσσάρων εξόδων στο δωμάτιο. Η εικόνα αυτή έχει παρθεί απ' τον Hengst (2002).

4.3.2 Αφαιρετικός Τρόπος Απλοποίησης Κατάστασης (State Abstraction)

Η συνάρτηση E του HexQ σαφώς πρέπει να περιέχει όλες τις καταστάσεις του ολικού MDP. Παρόλα αυτά ο αλγόριθμος αποθηκεύει τις καταστάσεις της κάθε περιοχής και όχι όλες τις καταστάσεις του ολικού MDP (όπου οι πρώτες είναι προφανώς πολύ λιγότερες απ' τον συνολικό αριθμό καταστάσεων). Δηλαδή για κάθε περιοχή δημιουργείται ένας πίνακας E με δυναμικές διαστάσεις (καταστάσεις της περιοχής^e x ενέργειες^e). Έστω S το σύνολο των καταστάσεων της περιοχής και S' το σύνολο όλων των καταστάσεων. Τότε με αυτή την μεθοδολογία, εξοικονομείται μνήμη κατά |S'|/|S|.

4.3.3 Συμπίεση Πολυδιάστατων Μαρκοβιανών Διαδικασιών Απόφασης

Θα δούμε λεπτομερώς στα επόμενα κεφάλαια τον αλγόριθμο HexQ και θα παρατηρήσουμε ότι για ΜΔΑ διάστασης >2 εισάγεται η έννοια της αναδρομής για να λυθεί το ολικό ΜΔΑ. Δηλαδή για πολυδιάστατα ΜΔΑ το ολικό ΜΔΑ λύνεται αναδρομικά στα επίπεδα ιεραρχίας παράγοντας ένα συμπαγή, άκυκλο, κατευθυνόμενο γράφο (*ArrayList Sub_Mdps*). Οπόταν στο δέντρο ιεραρχίας που παράγεται, σε κάθε επίπεδο, υπάρχει μία μεταβλητή μείον. Αυτό δείχνει την σύμπτυξη του ΜΔΑ αφού σε κάθε επίπεδο, σε κάθε αναδρομική κλήση που εκτελεί ο αλγόριθμος απουσιάζει η μεταβλητή του προηγούμενου επιπέδου. Οπόταν το sub-MDP (υπο-πρόβλημα) σε κάθε επίπεδο είναι σαφώς πιο απλουστευμένο απ' το προηγούμενο επίπεδο, όπως βλέπουμε και στο σχήμα 4.7. Επίσης οι κοινές πολιτικές που χρησιμοποιούνται από τα διάφορα υπο-προβλήματα για να εξέλθουν από τις περιοχές αποθηκεύονται μόνο μία φορά.



Σχήμα 4.7: Χωρίς να γνωρίζουμε (επιτηδευμένα) καθόλου λεπτομέρειες για το πρόβλημα του σχήματος παρατηρούμε ότι στο σχήμα απεικονίζεται ένα τρισδιάστατο ΜΔΑ. Στο 1^{ο} επίπεδο (το κατώτατο) υπάρχουν κοινά υπο-προβλήματα τα οποία επαναχρησιμοποιούνται απ' το επόμενο επίπεδο. Ο αλγόριθμος HexQ δημιουργεί αυτό το συμπαγή γράφο αφαιρώντας τα περιττά και διπλότυπα υπο-προβλήματα, δημιουργώντας ένα συμπαγή γράφο στον οποίο με την αναδρομική κλήση του αλγορίθμου λύνεται το πρόβλημα. Επίσης φαίνεται και αυτό που τονίστηκε στο 4.3.3, αν συγκρίνουμε για παράδειγμα τα sub-MDPs του κατώτατου επιπέδου με τα sub-MDPs του 2^{ου} επιπέδου, παρατηρούμε ότι τα sub-MDPs του 2^{ου} επιπέδου είναι πιο απλοποιημένα απ' τα sub-MDPs του 1^{ου}, διότι απουσιάζει μία μεταβλητή. Η εικόνα αυτή έχει παρθεί απ' τον Hengst (2002).

Κεφάλαιο 5

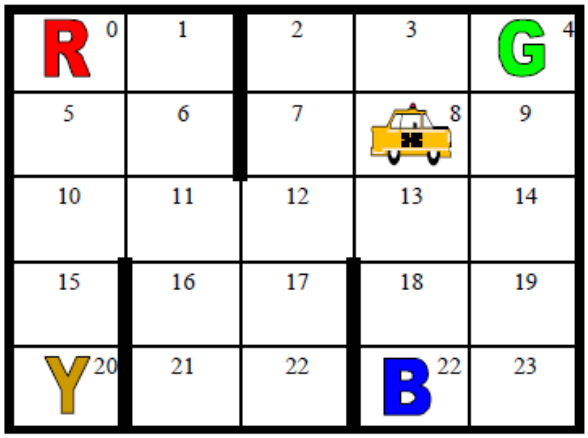
Ο αλγόριθμος HexQ και το «Πρόβλημα του Ταξί»

Σ’ αυτό το κεφάλαιο θα γίνει μια εισαγωγή στο πρόβλημα το οποίο εφαρμόστηκε ο ιεραρχικός αλγόριθμος ενισχυτικής μάθησης HexQ στα πλαίσια της εργασίας μου, το TP. Ακολούθως θα γίνει μια λεπτομερής ανάλυση του αλγορίθμου HexQ βήμα προς βήμα (σε σχέση με το συγκεκριμένο πρόβλημα), θα διασαφηνιστούν λεπτομέρειες της υλοποίησης αλλά θα επεξηγηθούν εκτενώς και οι δομές που χρησιμοποιήθηκαν στον αλγόριθμο. Γενικότερα, στόχος αυτού του κεφαλαίου είναι η κατανόηση της κάθε διαδικασίας του αλγορίθμου αλλά και το πώς ο αλγόριθμος αυτός αποσυνθέτει και λύνει ένα πρόβλημα αποδοτικά χωρίς την γνώση του μοντέλου.

5.1 Το πρόβλημα του ταξί

Πρόκειται να ασχοληθούμε με το TP όπου ένα ταξί πρέπει να παραλάβει τον επιβάτη από την αρχική θέση στην οποία βρίσκεται και να τον μεταφέρει στο προορισμό του σε ελάχιστο αριθμό βημάτων. Πρόκειται για ένα τρισδιάστατο επεισοδιακό παιχνίδι, όπου το ταξί και ο επιβάτης βρίσκονται σ’ ένα πλαίσιο 5X5, ο επιβάτης αρχικά βρίσκεται σε ένα απ’ τα τέσσερα χρωματισμένα κελιά, το ταξί μπορεί αρχικά να βρίσκεται σε οποιοδήποτε κελί και τελικός προορισμός του επιβάτη μπορεί να είναι ένα από τα τέσσερα χρωματισμένα κελιά. Σε κάθε βήμα

το ταξί μπορεί να εκτελέσει μία απ' τις έξι ενέργειες (N=↑, S=↓, E=→, W=←, GET, PUT). Σε περίπτωση που το ταξί χτυπήσει σε ένα απ' τα εμπόδια παραμένει στην ίδια θέση. Η αμοιβή για μια επιτυχή αποβίβαση του επιβάτη στον προορισμό είναι 20, ενώ για εκτέλεση ενέργειας GET ενώ ο επιβάτης δεν είναι εκεί ή εκτέλεση ενέργειας PUT ενώ ο επιβάτης δεν είναι στο ταξί, δίνουν -10. Για οποιαδήποτε άλλη περίπτωση το ταξί παίρνει αμοιβή -1 (όλες αυτές οι λεπτομέρειες καθορίζονται στο αρχείο ***singleSmall.txt*** που εισάγεται στο πρόγραμμα από τη γραμμή εντολών και ορίζει τις αμοιβές για κάθε ενέργεια, τα εμπόδια κλπ). Το παιχνίδι τελειώνει όταν το ταξί μεταφέρει με επιτυχία τον επιβάτη στον προορισμό του ή όταν ο μέγιστος αριθμός βημάτων έχει εξαντληθεί. Το σχήμα (5.1) αναπαριστά το TP γραφικά.



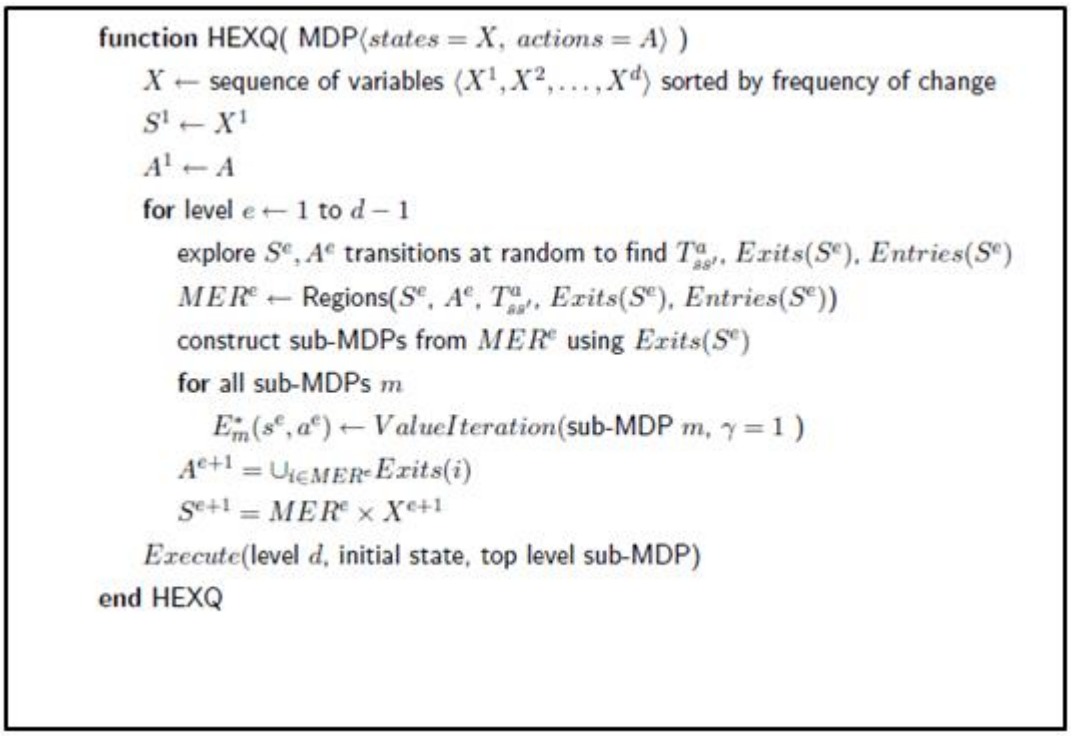
Σχήμα 5.1: Στο σχήμα φαίνονται το ταξί, τα χρωματισμένα γράμματα R, Y, G, B είναι οι πιθανοί προορισμοί που συγχρόνως αποτελούν και τις πιθανές αρχικές θέσεις του επιβάτη, ενώ σε όλο το πλαίσιο είναι οι πιθανές θέσεις του ταξί (και συγχρόνως το περιβάλλον στο οποίο επιτρέπεται να κινείται). Οι μαύρες κατακόρυφοι ράβδοι αποτελούν εμπόδια, τοίχους. Δηλαδή το ταξί που αποτελεί τον εκπαιδευόμενο, δεν μπορεί να κινηθεί από μία κατάσταση σε μια άλλη αν μεσολαβεί τοίχος. Η εικόνα αυτά έχει παρθεί απ' τον Hengst (2002).

Το TP αποτελεί πρόβλημα ΕΜ και χρησιμοποιήθηκε από τον Dietterich (2000) για την παρουσίαση της MAXQ ιεραρχικής διάσπασης αλλά υλοποιήθηκε και από τον Λάμπρου (2011). Κάπου εδώ να αναφέρω ότι η δική μου υλοποίηση περιλαμβάνεται στην υλοποίηση του Λάμπρου (2011) διότι αρχικός στόχος ήταν η επέκταση της εργασίας του. Εν τέλει χρησιμοποιήθηκαν δυο συναρτήσεις μόνο αλλά οι υπόλοιπες δεν χρησιμοποιήθηκαν, διότι τα προβλήματα ήταν ανεξάρτητα άρα δεν επεκτάθηκε.

Γενικός στόχος του προβλήματος είναι το ταξί να μάθει την τοποθεσία του επιβάτη, να τον παραλάβει και να τον μεταφέρει στο προορισμό του σε όσο το δυνατό λιγότερα βήματα. Άρα αναμένουμε με την χρήση του αλγόριθμου HexQ να επιταχύνεται η μάθηση στην πορεία της εκπαίδευσης και το πρόβλημα να λύνεται αποδοτικά σε ελάχιστο αριθμό βημάτων χωρίς ιδιαίτερη υπολογιστική πολυπλοκότητα και απαιτήσεις μνήμης.

5.2 HexQ

Στο σχήμα (5.2) παρουσιάζεται ο ψευδοκώδικας HexQ, του Hengst (2002). Στον ψευδοκώδικα καλούνται και άλλες συναρτήσεις σε μορφή ψευδοκώδικα που διατίθενται στον Hengst (2002) και φαίνονται σε αυτό το κεφάλαιο.



Σχήμα 5.2: Παρουσιάζεται ο αλγόριθμος HexQ (αντιστοιχεί στην κλάση HexQ *(ΠΑΡΑΡΤΗΜΑ Α, σελ. 1)* που υπάρχει στο παράρτημα Α). Παρατηρούμε ότι υπάρχουν κλήσεις συναρτήσεων τις οποίες θα δούμε στη συνέχεια. Ο αλγόριθμος λύνει ένα ΜΔΑ χωρίς να γνωρίζει το μοντέλο (συνάρτηση μεταβάσεων και συνάρτηση αμοιβής) όπως παρατηρούμε στα ορίσματα του. Και στην τελευταία του γραμμή λύνει αναδρομικά το πρόβλημα από πάνω προς τα κάτω (συνάρτηση *execute(d, s, 0) (δες ΠΑΡΑΡΤΗΜΑ Α, σελ. 36),* κλάση *HexQ (ΠΑΡΑΡΤΗΜΑ Α, σελ. 1)*). Ο ψευδοκώδικας έχει παρθεί απ' τον Hengst (2002).

5.2.1 Ταξινόμηση Μεταβλητών

Όπως αναφέραμε ένα πρόβλημα ενισχυτικής μάθησης χαρακτηρίζεται από ένα αριθμό μεταβλητών. Στο TP αυτές οι μεταβλητές απεικονίζονται στο σχήμα 5.3 και είναι οι εξής X= (θέση ταξί, θέση επιβάτη, προορισμός) (συνάρτηση `initializeX()` *(δες ΠΑΡΑΡΤΗΜΑ Α, σελ. 25)*). Υπάρχουν δηλαδή τρεις μεταβλητές άρα d=3 (`setDepth()` *(δες ΠΑΡΑΡΤΗΜΑ Α, σελ. 5)*).

Μεταβλητές του TP
Θέση ταξί (Taxi Location)
Θέση Επιβάτη (Passenger Location)
Προορισμός (Destination)

Σχήμα 5.3: Στον πίνακα φαίνονται ταξινομημένες σε φθίνουσα σειρά οι τρεις μεταβλητές του προβλήματος ανάλογα με την συχνότητα αλλαγής τους (*ArrayList X*).

Ο αλγόριθμος σε αυτό το στάδιο εκτελεί μια τυχαία εξερεύνηση (τα βήματα ορίζονται από τον χρήστη όσο αφορά το τέλος της εξερεύνησης) με στόχο να παρατηρήσει ποιες μεταβλητές αλλάζουν συχνότερα και ποιες όχι. Στο συγκεκριμένο πρόβλημα η μεταβλητή που αλλάζει συχνότερα είναι η *θέση ταξί*, ακολουθεί η μεταβλητή *θέση επιβάτη* και τέλος η μεταβλητή *προορισμός* η οποία δεν αλλάζει τιμή στα πλαίσια της εξερεύνησης. Τα αποτελέσματα είναι απολύτως λογικά διότι το ταξί μέχρι να εντοπίσει τον επιβάτη, να τον παραλάβει και να τον αφήσει στο προορισμό αλλάζει συνεχώς θέση. Ενώ ο επιβάτης δεν αλλάζει τόσο συχνά θέση. Ο λόγος που γίνεται αυτή η ταξινόμηση είναι για να χτιστεί η ιεραρχία ορθά (ο HexQ αποτελεί ένα ιεραρχικό αλγόριθμο). Στο κατώτατο σημείο ιεραρχίας βρίσκεται η μεταβλητή που αλλάζει συχνότερα τιμή, η *θέση ταξί*, στο $2^{ο}$ επίπεδου η μεταβλητή *θέση επιβάτη* και στο $3^{ο}$ και τελευταίο επίπεδο βρίσκεται η μεταβλητή *προορισμός*. Αυτή η διαδικασία είναι το $1^{ο}$ βήμα του αλγόριθμου στο σχήμα 5.2.

5.2.2 Αρχικοποιήσεις

Στο βήμα 2 και 3 του σχήματος 5.2 γίνονται οι απαραίτητες αρχικοποιήσεις πριν την έναρξη του βρόγχου στη γραμμή 4 (`initializeS(e)` *) (δες ΠΑΡΑΡΤΗΜΑ Α, σελ. 25)*, `initializeA()` *(δες ΠΑΡΑΡΤΗΜΑ Α, σελ. 25)*). Αυτές οι αρχικοποιήσεις αφορούν το σύνολο ενεργειών (A, *ArrayList Primitve_Actions*) του τρέχοντος επιπέδου (επίπεδο 1, κατώτατο) και το σύνολο καταστάσεων (S)(*ArrayList States*) του ίδιου επιπέδου. Εφ' όσον βρισκόμαστε στο κατώτατο επίπεδο αναφερόμαστε σε primitive ενέργειες και πιο συγκεκριμένα αναφερόμαστε στις ενέργειες του ταξί. Το ταξί μπορεί να κινηθεί πάνω, κάτω, δεξιά, αριστερά, μπορεί να παραλάβει και να αφήσει τον επιβάτη. Δηλαδή το σύνολο A^1={North, South, East, West, GET, PUT}. Φυσικά οι έξι ενέργειες είναι κωδικοποιημένες με έξι ακέραιους αριθμούς (0…5) (*ArrayList Actions*). Όσο αφορά το σύνολο καταστάσεων στο $1^ο$ επίπεδο, αναφέραμε και πιο πάνω ότι το ταξί κινείται σε ένα περιβάλλον 5 x 5, άρα αποτελείται από εικοσιπέντε διαφορετικές καταστάσεις και S^1={0…24}.

5.2.3 Εξερεύνηση συνάρτησης μεταβάσεων, εξόδων, εισόδων, καταστάσεων και ενεργειών.

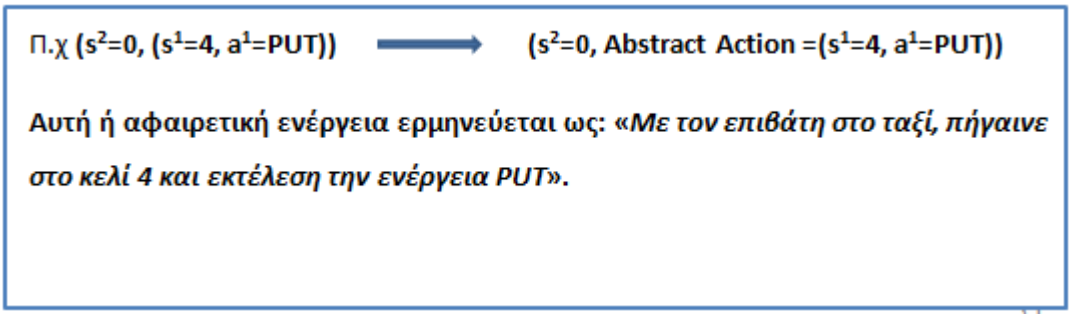
Προαναφέρθηκε στο 5.2.2 ότι ο πράκτορας εκτελεί μια τυχαία εξερεύνηση με στόχο να παρατηρήσει τη συχνότητα αλλαγής της κάθε μεταβλητής. Δεν παρατηρεί μόνο αυτό, αλλά εξερευνά και το περιβάλλον (με στόχο να μάθει το άγνωστο μοντέλο).

Ανακαλύπτει τα συνάρτηση μεταβάσεων, το σύνολο εξόδων και εισόδων, τις καταστάσεις και τις ενέργειες του τρέχοντος επιπέδου e που βρίσκεται. Με βάση τους ορισμούς που δόθηκαν στο προηγούμενο κεφάλαιο, ακολουθούν τα ευρήματα του HexQ για το επίπεδο ένα (στον $1^ο$ κύκλο του βρόγχου):

ΕΞΟΔΟΙ (για e=1)	ΕΞΟΔΟΙ (για e=2)
(s^1=0, a^1=GET)	
(s^1=4, a^1=GET)	
(s^1=20, a^1=GET)	
(s^1=23, a^1=GET)	
(s^1=0, a^1=PUT)	**(s^2=0, (s^1=0, a^1=PUT))**
(s^1=4, a^1=PUT)	**(s^2=0, (s^1=4, a^1=PUT))**
(s^1=20, a^1=PUT)	**(s^2=0, (s^1=20, a^1=PUT))**
(s^1=23, a^1=PUT)	**(s^2=0, (s^1=23, a^1=PUT))**

Σχήμα 5.4: Οι έξοδοι των δυο επιπέδων είναι αποθηκευμένες στο *ArrayList Exits* και βρέθηκαν μετά από κλήση της συνάρτησης *findExits(e) (δες ΠΑΡΑΡΤΗΜΑ Α, σελ. 8).*

Στο 1^{ο} επίπεδο ο HexQ ανακαλύπτει οκτώ εξόδους (όπως βλέπουμε στο σχήμα 5.4) και στο 2^{ο} επίπεδο τέσσερις. Το S^e είναι η κατάσταση του επιπέδου e και το a^e η ενέργεια του επιπέδου e (Οι καταστάσεις του προβλήματος στο *ArrayList States* και οι ενέργειες στο *ArrayList Actions* αλλά είναι και χωρισμένες στα *ArrayLists Abstract_Actions & Primitive_Actions* αναλόγως του επιπέδου ιεραρχίας που ανήκουν). Όπως προαναφέραμε στο προηγούμενο κεφάλαιο στον ορισμό των εξόδων, ορίσαμε ως έξοδο ένα ζεύγος (κατάσταση, ενέργεια), όμως παρατηρούμε ότι στο επίπεδο 2 η δομή της εξόδου διαφέρει. Και αυτό γιατί (πέραν του 1^{ου} επιπέδου) οι ενέργειες δεν είναι primitive αλλά αφαιρετικές. Μια αφαιρετική ενέργεια έχει την ίδια δομή με μία έξοδο, δηλαδή ζεύγος (κατάσταση, ενέργεια) όπως φαίνεται στο σχήμα (5.5). Οι έξοδοι των επιπέδων βρίσκονται με τη συνάρτηση `findExits(e)` *(δες ΠΑΡΑΡΤΗΜΑ Α, σελ. 8)*.



Σχήμα 5.5: Παράδειγμα αφαιρετικής ενέργειας. Όλες οι αφαιρετικές ενέργειες αποθηκεύτηκαν στο ArrayList Abstract_Actions και βρέθηκαν μετά από κλήση της συνάρτησης *nextLevelAbstractActions(e)) (δες ΠΑΡΑΡΤΗΜΑ Α, σελ. 24).* αφού κληθεί η *findExits(e) (δες ΠΑΡΑΡΤΗΜΑ Α, σελ. 8).*

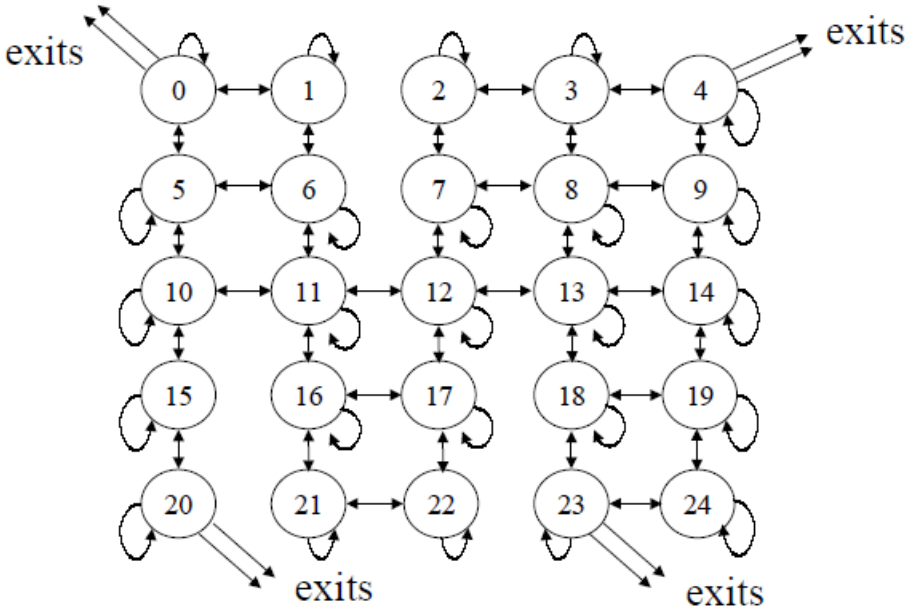
Στο 1^{ο} επίπεδο βρέθηκαν οκτώ έξοδοι διότι από τα τέσσερα κελιά (0, 4, 20, 23) που αποτελούν τις πιθανές αρχικές θέσεις του επιβάτη, αν εκτελεστούν οι ενέργειες GET ή PUT τότε αλλάζει τιμή η μεταβλητή 2, η μεταβλητή *θέση επιβάτη*. Δηλαδή σε αυτά τα τέσσερα κελιά αν εκτελεστεί GET ο επιβάτης θα βρεθεί στο ταξί (άρα θα αλλάξει θέση ο επιβάτης), ενώ αν εκτελεστεί PUT τότε ο επιβάτης θα κατέβει από το ταξί άρα και πάλι θα αλλάξει θέση. Αυτή η συνθήκη είναι ένας από τους ορισμούς των εξόδων που δώσαμε στο προηγούμενο κεφάλαιο.

Στο 2^{ο} επίπεδο υπάρχουν τέσσερις έξοδοι, όπου και οι τέσσερις προϋποθέτουν σε αυτό το επίπεδο ο επιβάτης να βρίσκεται στο ταξί (s^2=0). Είναι έξοδοι για διότι αν ο επιβάτης βρίσκεται στο ταξί και εκτελεστεί PUT σε ένα από τους τέσσερις προορισμούς (0, 4, 20, 23) το πρόβλημα θα τερματιστεί. Αυτή η συνθήκη είναι επίσης ένας από τους ορισμούς των εξόδων που δώσαμε στο προηγούμενο κεφάλαιο. Όλες οι είσοδοι αποθηκεύτηκαν στο *ArrayList Entries* μετά από κλήση της συνάρτησης *findEntries(e) (δες ΠΑΡΑΡΤΗΜΑ Α, σελ. 10).*

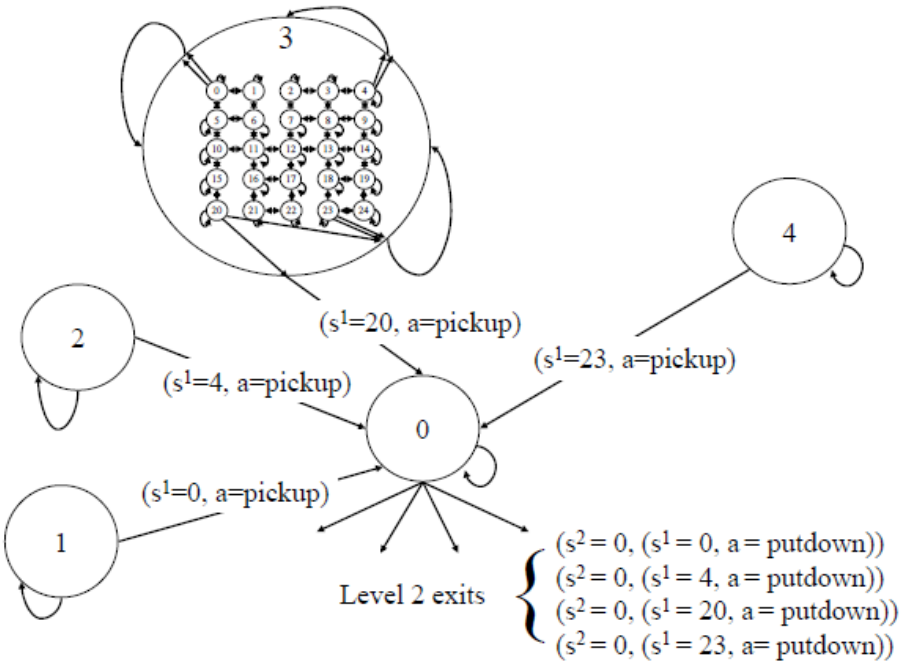
ΕΙΣΟΔΟΙ (για e=1)	ΕΙΣΟΔΟΙ (για e=2)
$S^1=0$ $S^1=5$ $S^1=10$ $S^1=15$ $S^1=20$	$S^2=0$
$S^1=1$ $S^1=6$ $S^1=11$ $S^1=16$ $S^1=21$	$S^2=4$
$S^1=2$ $S^1=7$ $S^1=12$ $S^1=17$ $S^1=22$	$S^2=20$
$S^1=3$ $S^1=8$ $S^1=13$ $S^1=18$ $S^1=23$	$S^2=23$
$S^1=4$ $S^1=9$ $S^1=14$ $S^1=19$ $S^1=24$	

Σχήμα 5.6: Οι είσοδοι των δυο επιπέδων (*ArrayList Entries*).

Οι είσοδοι αποτελούν τις αρχικές θέσεις της μεταβλητής άρα στο επίπεδο 1 οι αρχικές θέσεις του ταξί είναι όλα τα κελιά αφού η θέση ταξί καθορίζεται τυχαία. Σε αντίθεση με τις αρχικές θέσεις του 2ου επιπέδου όπου ο επιβάτης μπορεί να έχει αρχικές θέσεις του τέσσερις προορισμούς αλλά όχι αρχική θέση μέσα στο ταξί. Το σύνολο των εισόδων για τα δυο επίπεδα φαίνεται στο σχήμα (5.6). Οι είσοδοι των επιπέδων βρίσκονται με τη συνάρτηση *findEntries(e) (δες ΠΑΡΑΡΤΗΜΑ Α, σελ. 10)*. Ακολουθούν τα μοντέλα μεταβάσεων των δυο επιπέδων στο σχήμα (5.7) για το 1ο επίπεδο και στο σχήμα (5.8) για το 2ο επίπεδο. Η συνάρτηση μεταβάσεων βρίσκεται μέσω της συνάρτησης *findTransitionModel(SA_Gridworld.size_x, SA_Gridworld.size_y,SA_Gridworld.wall, e) (δες ΠΑΡΑΡΤΗΜΑ Α, σελ. 5).*



Σχήμα 5.7: Η συνάρτηση μεταβάσεων του επιπέδου 1 μοντελοποιημένη. Απεικονίζονται οι οκτώ έξοδοι του 1^{ου} επιπέδου. Σ' αυτό το μοντέλο κινείται το ταξί, μέχρι να επιβιβάσει τον επιβάτη στο ταξί ή μέχρι να τον αποβιβάσει σε ένα απ' τους τέσσερις προορισμούς. Σε αυτό το περιβάλλον εκτελούνται οι primitive ενέργειες του 1^{ου} επιπέδου (North, South, East, West, Get, Put). Η εικόνα αυτή έχει παρθεί απ' τον Hengst (2002).



Σχήμα 5.8: Η συνάρτηση μεταβάσεων του επιπέδου 2 μοντελοποιημένη. Απεικονίζονται και οι τέσσερις έξοδοι του 2^{ου} επιπέδου. Ο κόμβος 0 στο 2^{ο} επίπεδο αντικατοπτρίζει τον επιβάτη μέσα στο ταξί, ενώ οι κόμβοι 1…4 τις τέσσερις περιοχές-προορισμούς (0, 4, 20, 23) του επιπέδου ένα που προκλήθηκε έξοδος (γι' αυτό στο εσωτερικό του κόμβου τρία φαίνεται το προηγούμενο επίπεδο). Κάτι πολύ σημαντικό που πρέπει να γίνει αντιληπτό απ' το σχήμα είναι ότι εάν απ' το 1^{ο} επίπεδο και τις οκτώ εξόδους δεν επιλεχθούν οι έξοδοι που εμπλέκουν την ενέργεια GET ((s^1=0, a^1=GET), (s^1=4, a^1=GET), (s^1=20, a^1=GET), (s^1=23, a^1=GET)), τότε δεν επιβιβάζεται ο επιβάτης στο ταξί άρα δεν γίνεται μετάβαση στο κόμβο μηδέν του 2^{ου} επιπέδου (που συμβολίζει τον επιβάτη στο ταξί) και παραμένει το πρόβλημα στο 1^{ο} επίπεδο καθώς και η θέση επιβάτη παραμένει ίδια.

Η μετάβαση απ' τον ένα κόμβο στον άλλο γίνεται μέσω αφαιρετικών ενεργειών διότι δεν βρισκόμαστε στο κατώτατο επίπεδο ιεραρχίας. Η εικόνα αυτή έχει παρθεί απ' τον Hengst (2002).

5.2.4 Εύρεση Περιοχών

Οι Μαρκοβιανές Ισοδύναμες Περιοχές (Markov Equivalent Regions, MERS) είναι από τις σημαντικότερες δομές του αλγόριθμου HexQ. Η συνάρτηση *Regions(e) (δες ΠΑΡΑΡΤΗΜΑ Α, σελ. 13)* βρίσκει τις περιοχές των επιπέδων και είναι η αντίστοιχη του ψευδοκώδικα της Regions του Hengst (2002).Στο σχήμα (5.9) φαίνεται ο ψευδοκώδικας της συνάρτησης Regions που βρίσκει τις περιοχές του κάθε επιπέδου. Ακολούθως στο σχήμα (5.10) φαίνεται ο αλγόριθμος SCC (συνάρτηση *SCC(int states, int e) (δες ΠΑΡΑΡΤΗΜΑ Α, σελ. 26)* είναι η αντίστοιχη του ψευδοκώδικα SCC του Hengst (2002)) που βρίσκει αν ο γράφος είναι ισχυρά συνδεδεμένος ή όχι και από πόσους ισχυρά συνδεδεμένους κόμβους αποτελείται (η συνάρτηση SCC χρησιμοποιείται από τη συνάρτηση Regions).

```
function Regions( states S^e, actions A^e, T^a_{ss'}, Exits(S^e), Entries(S^e))
    // Find SCCs so that all states are reachable from all others
    repeat until number of SCCs do not increase
        for each s, s' ∈ S^e and a ∈ A^e (where s transitions to s' on action a)
            if (T^a_{ss'} > 0 and {(s,a)} ⊈ Exits(S^e)) adj[s][s'] ← true
            else adj[s][s'] ← false
        SCC( states S^e, adj[s][s'] )
        for each s, a, s' connecting two SCCs
                adj[s][s'] ← false
                Exits(S^e) ← Exits(S^e) ∪ {(s,a)}
                Entry(S^e) ← Entries(S^e) ∪ {s'}
    end repeat
    // Join SCCs into regions so that all region entry states can reach all exit states
    for each s, a, s' connecting two SCCs
            if (SCC[s] has no other Exit or SCC[s'] has no other Entry
                Exits(S^e) ← Exits(S^e) − {(s,a)}
                Entry(S^) ← Entries(S^e) − {s'}
                for each s^i if(s^i = SCC[s'])
                    SCC[s^i] ← SCC[s]
    return |SCCs|, SCC[·], Exits(S^e), Entries(S^e)
end Regions
```

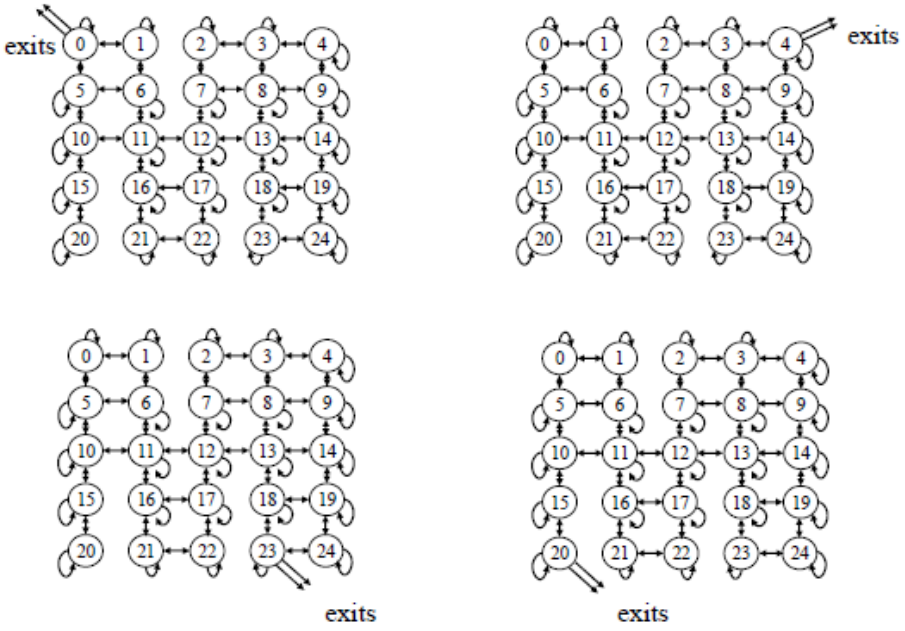
Σχήμα 5.9: Η συνάρτηση Regions βρίσκει των αριθμό των περιοχών του επιπέδου που την καλεί. Και στο 1^{ο} αλλά και στο 2^{ο} επίπεδο η Regions βρίσκει μόνο μία περιοχή. Αφού στο 1^{ο} επίπεδο ο γράφος είναι ισχυρά συνδεδεμένος, ένα MER δημιουργείται (διότι απ' όλες τις αρχικές θέσεις μπορεί να γίνει μετάβαση σε καταστάσεις εξόδου χωρίς αλλαγή περιοχής). Στο 2^{ο} επίπεδο υπάρχουν πέντε διαφορετικά SCCs (0…4) τα οποία συγχωνεύονται για να δημιουργήσουν μόνο ένα MER (αφού απ' όλες τις αρχικές θέσεις μπορεί και πάλι να γίνει μετάβαση στις καταστάσεις εξόδου χωρίς να εξέλθει της περιοχής). Ο αλγόριθμος έχει παρθεί απ' τον Hengst (2002). Για να βρεθεί ο πίνακας γειτνίασης (adj[][]) που χρησιμοποιείται δημιουργήθηκε η συνάρτηση *findAdjacentMatrix1(SA_Gridworld.size_x, SA_Gridworld.size_y, SA_Gridworld.wall) (δες ΠΑΡΑΡΤΗΜΑ Α, σελ. 10)* για το επίπεδο 1 και findAdjacentMatrix2(e) *(δες ΠΑΡΑΡΤΗΜΑ Α, σελ. 11)* για το επίπεδο 2. Επίσης ο τελευταίος βρόγχος του ψευδοκώδικα αντιστοιχεί στη συνάρτηση *Join_Sccs(e) (δες ΠΑΡΑΡΤΗΜΑ Α, σελ. 18)* και ο 2ος βρόγχος του ψευδοκώδικα στη συνάρτηση *Connected_Sccs(e) (δες ΠΑΡΑΡΤΗΜΑ Α, σελ. 20).*

```
function SCC( states S, adj[s][s'] )
    initialise finTime ← 0
    initialise SCClable ← 0
    for each state s ∈ S
        initialise col[s] ← WHITE
        initialise f[s] ← undefined
        initialise SCC[s] ← undefined
    for each state s ∈ S if (col[s] = WHITE) DFS1(s)
    for each state s ∈ S col[s] ← WHITE
    for each state s ∈ S in order of decreasing f[s]
        if(col[s] = WHITE) DFS2(s)
        increment SCClabel
    return SCC[·], SCClabel

    DFS1(s)
        col[s] ← GRAY
        increment finTime
        for each state s' ∈ S if (adj[s][s'] and col[s'] = WHITE) DFS1(s')
        col[s] ← BLACK
        f[s] ← finTime
        increment finTime
    return

    DFS2(s)
        col[s] ← GRAY
        for each state s' ∈ S if (adj[s'][s] and col[s'] = WHITE) DFS2(s')
        col[s] ← BLACK
        SCC[s] ← SCClabel
    return

end SCC
```

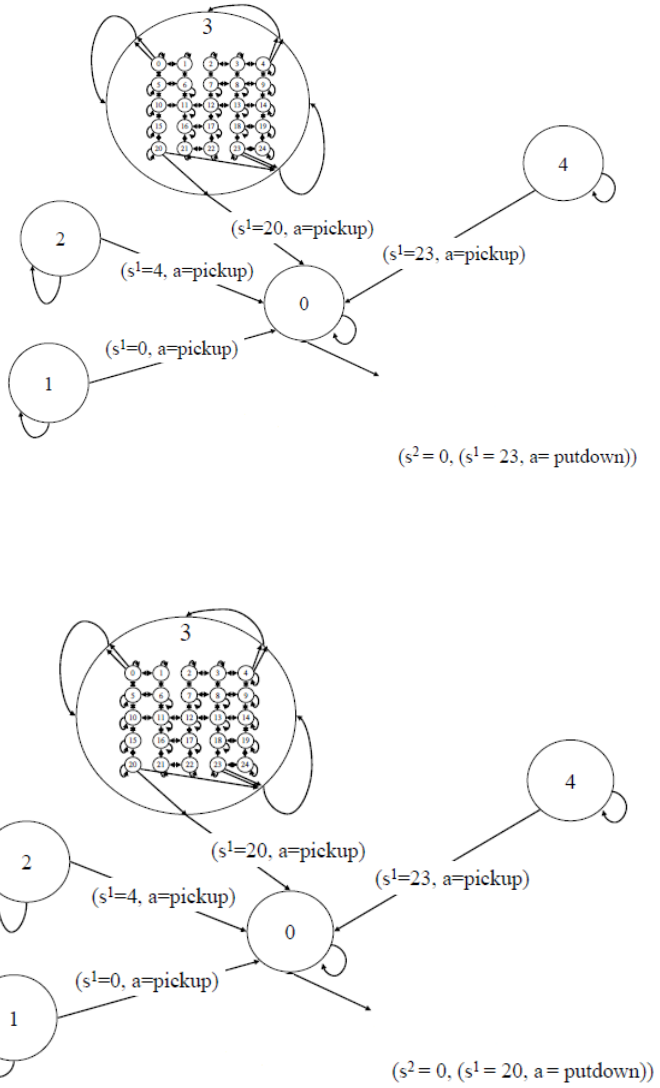
Σχήμα 5.10: Η συνάρτηση SCC επιστρέφει τον αριθμό των strongly connected components *(SCC label*) και τον *πίνακα SCC[]* μεγέθους όσο και ο αριθμός των καταστάσεων του επιπέδου e, που περιέχει την πληροφορία σε ποιο strongly connected component ανήκει η κάθε κατάσταση s. Στο 1^{o} επίπεδο ο γράφος στο σχήμα (5.7) είναι ένα SCC, ενώ στο 2^{o} επίπεδο (σχήμα 5.8) βρίσκει πέντε SCCs. Τα Sccs έχουν την ιδιότητα ότι όλοι οι κόμβοι πρέπει να επικοινωνούν με πιθανότητα 1. Ο αλγόριθμος έχει παρθεί απ' τον Hengst (2002). Οι συνάρτήσεις που αντιστοιχούν σε αυτόν τον ψευδοκώδικα είναι η *SCC(int states, int e) (δες ΠΑΡΑΡΤΗΜΑ Α, σελ. 26), DFS1(int s, int states) (δες ΠΑΡΑΡΤΗΜΑ Α, σελ. 24), DFS2(int s, int states) (δες ΠΑΡΑΡΤΗΜΑ Α, σελ. 24).*

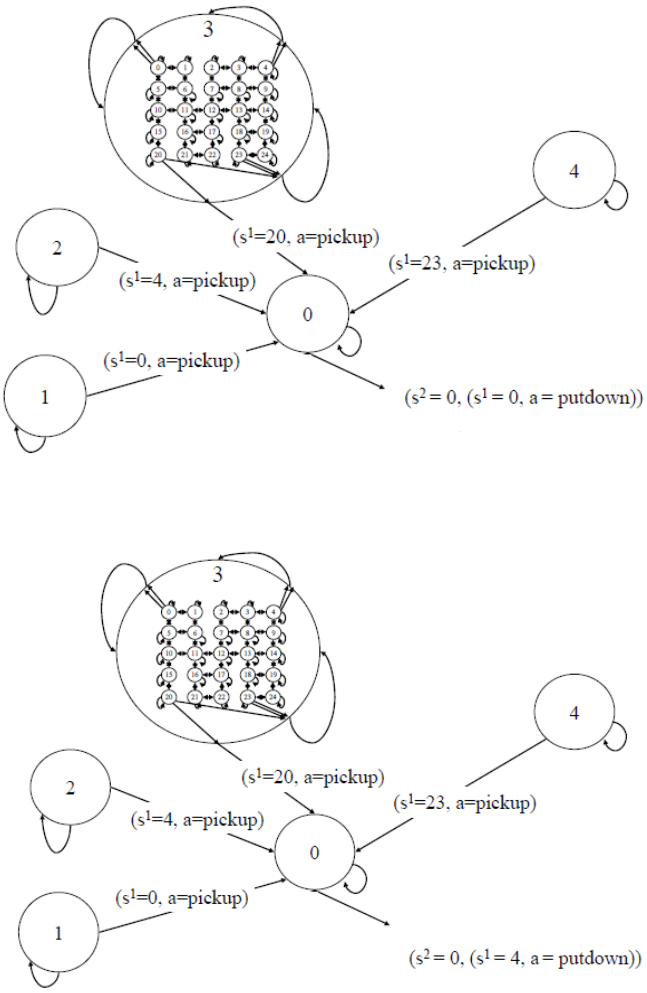
5.2.5 ΕΥΡΕΣΗ Sub-MDPs

Ακόμη μία εξ' ίσου σημαντική δομή που δημιουργεί ο αλγόριθμος HexQ αυτόματα (η συνάρτηση που το κάνει αυτό είναι η *constructSubMdps(e) (δες ΠΑΡΑΡΤΗΜΑ Α, σελ. 27) & constructTopLevelSub(d) (δες ΠΑΡΑΡΤΗΜΑ Α, σελ. 30)).* Όπως αναφέραμε και στο προηγούμενο κεφάλαιο σε κάθε επίπεδο υπάρχει ένας αριθμός από sub-MDPs σε κάθε περιοχή, τόσα όσα και ο αριθμός των καταστάσεων που προκαλούν έξοδο της περιοχής. Ακολουθούν τα sub-MDPs του $1^{ου}$ και $2^{ου}$ επιπέδου (σχήμα 5.11, 5.12).



Σχήμα 5.11: Τα τέσσερα sub-MDPs του 1^{ου} επιπέδου (για τη μοναδική περιοχή που βρέθηκε), όσα και οι καταστάσεις που προκαλούν έξοδο απ' το επίπεδο. Για κάθε sub-MDP ισχύει η ίδια συνάρτηση μεταβάσεων με τη μόνη διαφορά ότι για κάθε sub-MDP αντιστοιχεί μόνο μία κατάσταση που προκαλεί έξοδο με μία ή περισσότερες ενέργειες (δεν επηρεάζει τη δομή του sub-MDP αν μέσω μιας κατάστασης οδηγούν σε έξοδο περισσότερες από μια ενέργειες-γι' αυτό και τα διπλά βέλη στις εξόδους των sub-MDPs, διότι δυο ενέργειες οδηγούν απ' εκείνη την κατάσταση σε έξοδο). Η εικόνα είναι παραλλαγή του σχήματος (5.8) που έχει παρθεί απ' τον Hengst (2002).





Σχήμα 5.12: Τα τέσσερα sub-MDPs του 2ου επιπέδου (για τη μοναδική περιοχή που βρέθηκε), όσα και οι καταστάσεις που προκαλούν έξοδο απ' το επίπεδο. Για κάθε sub-MDP ισχύει η ίδια συνάρτηση μεταβάσεων με τη μόνη διαφορά ότι για κάθε sub-MDP αντιστοιχεί μόνο μία κατάσταση που προκαλεί έξοδο με μία αφαιρετική ενέργεια. Η εικόνα έχει παρθεί απ' τον Hengst (2002).

Το κάθε sub-MDP συνδέεται με μία και μόνο κατάσταση εξόδου και έχει ως υποστόχο να εξέλθει απ' αυτή την έξοδο. Γι' αυτό στη γραμμή 7 του HexQ (σχήμα 5.2), για την δημιουργία των sub-MDPs είναι απαραίτητη η γνώση των εξόδων του τρέχοντος επιπέδου και των περιοχών του επιπέδου. Επειδή στο βρόγχο υπάρχει η

διαδικασία δημιουργίας των sub-MDPs και ο βρόγχος στη συγκεκριμένη περίπτωση αφορά μόνο e=1, e=2, δεν διευκρινίζεται πότε δημιουργείται το sub-MDP του ανώτατου επιπέδου (*constructTopLevelSub(d) (δες ΠΑΡΑΡΤΗΜΑ Α, σελ. 30)*), παρόλα αυτά δημιουργήθηκε και βρίσκεται στο σχήμα (5.14). Επίσης μαζί με το κάθε sub-MDP συνδέεται ένας δισδιάστατος *πίνακας Ε* (διαστάσεων [αρ.καταστάσεων^e][αρ.ενεργειών^e], όπου στη συνέχεια αρχικοποιείται και χρησιμοποιείται για την εύρεση της βέλτιστης συνάρτησης $E^*_m(s,a)$ για κάθε sub-MDP, όπου είναι η μέγιστη αναμενόμενη συσσωρευμένη τιμή που λαμβάνει ο πράκτορας απ' την κατάσταση s με ενέργεια a, μέχρι την έξοδο του απ' το συγκεκριμένο sub-MDP m. Αφού βρεθούν τα sub-MDPs των περιοχών του τρέχοντος επιπέδου e, ο αλγόριθμος καλείται να βρει την βέλτιστη λύση για το καθένα απ' αυτά μέσω του αλγόριθμου Value Iteration που φαίνεται στο σχήμα (5.13).

function ValueIteration(MDP$\langle S, A, T, R \rangle, \gamma$)
  initialise $Q(s,a) \leftarrow 0$
  repeat until $\triangle <$ small positive number
    $\triangle \leftarrow 0$
    for each state $s \in S$
      for each action $a \in A$
        $q \leftarrow Q(s,a)$
        $Q(s,a) \leftarrow \sum_{s'} T^a_{ss'}[R^a_{ss'} + \gamma V(s')]$
        $\triangle \leftarrow \max(\triangle, |q - Q(s,a)|)$
  end repeat
  return $Q(s,a)$
end ValueInteration

Σχήμα 5.13: Η συνάρτηση ValueIteration λαμβάνει ως όρισμα ένα sub-MDP και την τιμή του ρυθμού έκπτωσης γ και επιστρέφει ένα πίνακα Q. Για το επίπεδο 1, η συνάρτηση αυτή λειτουργεί όμοια με τον αλγόριθμο Q-Learning (Watkins, 1989) ο οποίος βρίσκει την καταλληλότερη

συνάρτηση ενέργειας –αξίας. Ο αλγόριθμος έχει παρθεί απ' τον Hengst (2002). Η συνάρτηση *valueIteration(sub_mdp m, State_Variables state, int e) (δες ΠΑΡΑΡΤΗΜΑ Α, σελ. 33)* υλοποιήθηκε για τον ομώνυμο ψευδοκώδικα.

5.2.6 Εύρεση Καταστάσεων και Ενεργειών του επόμενου επιπέδου e+1

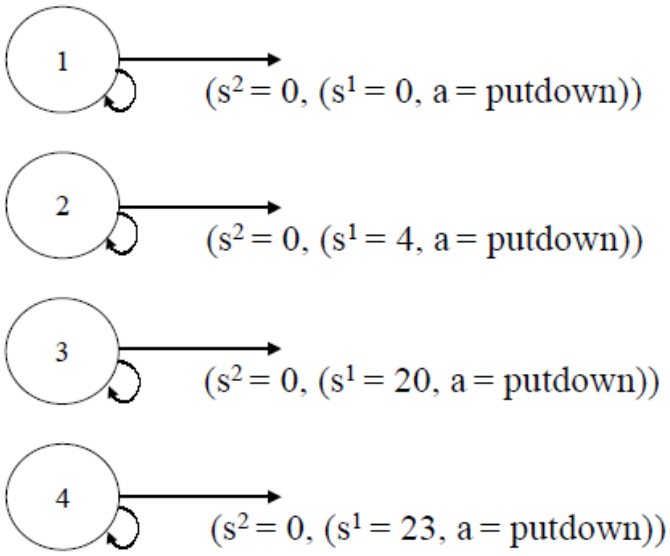
Κάπου εδώ ο αλγόριθμος στις γραμμές 10 και 11 (σχήμα 5.2), γνωρίζοντας τις εξόδους των περιοχών του τρέχοντος επιπέδου e, τον αριθμό των περιοχών του τρέχοντος επιπέδου e και το σύνολο καταστάσεων του επόμενου επιπέδου e+1 (το σύνολο καταστάσεων της επόμενης μεταβλητής, της μεταβλητής με την αμέσως λιγότερη συχνότητα αλλαγής) βρίσκει τις καταστάσεις και τις ενέργειες του επόμενου επιπέδου e+1 (συνάρτηση *nextLevelAbstractActions(e)) (δες ΠΑΡΑΡΤΗΜΑ Α, σελ. 24), initializeS(e + 1)) (δες ΠΑΡΑΡΤΗΜΑ Α, σελ. 25))*.

Δηλαδή οι ενέργειες του επόμενου επιπέδου (e+1) αποτελούν την ένωση των εξόδων του τρέχοντος επιπέδου (e) και όλων των περιοχών του επιπέδου αυτού (e). Γι' αυτό και το σύμβολο της ένωσης στη γραμμή 10 του αλγορίθμου. Δηλαδή οι ενέργειες του επιπέδου 1 είναι σαφώς οι 6 ενέργειες (0...5) που ορίστηκαν εξ' αρχής στην γραμμή 3 στου αλγορίθμου, ενώ οι αφαιρετικές ενέργειες του επιπέδου 2, αποτελούν τις 8 εξόδους του επιπέδου 1 ((s^1=0, a^1=GET), (s^1=4, a^1=GET), (s^1=20, a^1=GET), (s^1=23, a^1=GET), (s^1=0, a^1=PUT), (s^1=4, a^1=PUT), (s^1=20, a^1=PUT), (s^1=23, a^1=PUT)). Παρομοίως στο επίπεδο 3 οι αφαιρετικές ενέργειες αποτελούν τις εξόδους του προηγούμενου επιπέδου, δηλαδή τις 4 εξόδους του επιπέδου 2 ((s^2=0, (s^1=0, a^1=PUT)), (s^2=0, (s^1=4, a^1=PUT)), (s^2=0, (s^1=20, a^1=PUT)), (s^2=0, (s^1=23, a^1=PUT))).

Ενώ οι καταστάσεις του επόμενου επιπέδου βρίσκονται απ' το γινόμενο του αριθμού των περιοχών επί το σύνολο καταστάσεων της επόμενης μεταβλητή στην ιεραρχία. Αλλά επειδή στο συγκεκριμένο πρόβλημα (το TP) που εφαρμόσαμε τον HexQ, σε κάθε επίπεδο βρέθηκε μόνο μία περιοχή, οι καταστάσεις του επόμενου επιπέδου αποτελούν μόνο το σύνολο καταστάσεων της επόμενης μεταβλητής. Δηλαδή για e=1 η μεταβλητή X^1=*θέση ταξί*, οι καταστάσεις είναι 25 και ορίζονται στη γραμμή 2 του αλγόριθμου HexQ (σχήμα 5.2). Για e=2, X^2=*θέση επιβάτη* και οι καταστάσεις θα είναι 5 (σχήμα 5.8) ενώ για e=3, X^3=*προορισμός,* οι καταστάσεις του τρίτου επιπέδου θα είναι 4 (σχήμα 5.13).

5.2.7 Συνάρτηση Execute

Η συνάρτηση Execute (σχήμα 5.15) εκτελείται ουσιαστικά όταν τερματίσει ο βρόγχος (γραμμή 4-11) για να λύσει το ανώτατο sub-MDP (άρα το τρέχον επίπεδο e ισούται με τρία και συσχετίζεται με τη μεταβλητή προορισμός) και υλοποιήθηκε η ομώνυμη συνάρτηση *execute(d, s, 0) (δες ΠΑΡΑΡΤΗΜΑ Α, σελ. 36)*. Εκτελείται αναδρομικά για κάθε επίπεδο καλώντας τον εαυτό της,



Σχήμα 5.14: Αποτελεί το μοναδικό sub-MDP του ανώτατου επιπέδου, για e=3 που αφορά την 3η μεταβλητή *προορισμός (constructTopLevelSub(d) (δες ΠΑΡΑΡΤΗΜΑ Α, σελ. 30)).* Η εικόνα έχει παρθεί απ' τον Hengst (2002).

με πολιτικές κατώτερων επιπέδων, ξεκινώντας από το ψηλότερο επίπεδο (d=3). Αυτή η συνάρτηση οδηγεί στη λύση του προβλήματος. Λαμβάνει ως όρισμα το τρέχον επίπεδο (δηλαδή d=3), μια αρχική κατάσταση και ένα sub-MDP (τον αριθμό που αντιστοιχεί στο συγκεκριμένο sub-MDP). Το 3ο όρισμα, ο αριθμός αυτός ενός sub-MDP συνδέεται με μια primitive ή abstract ενέργεια. Άρα όταν επιστρέφει η Execute και ενώ έχει εκτελέσει μια primitive ή abstract ενέργεια, ανανεώνει έναν πίνακα E (αυτόν που επέστρεψε ο αλγόριθμος Value Iteration υποκεφάλαιο 5.2.4) με την βοήθεια της συνάρτησης value (σχήμα 5.16) (συνάρτηση *Value(int e, sub_mdp m, State_Variables s) (δες ΠΑΡΑΡΤΗΜΑ Α, σελ. 35)).* Την 1η φορά όμως καλείται με το ανώτατο μοναδικό sub-MDP του επιπέδου 3, με τον αριθμό μηδέν (δηλαδή αφαιρετική ενέργεια μηδέν). Η κλήση της είναι την 1η φορά: Execute (*top level, starting state, 0)*. Όταν η συνάρτηση Execute τερματιστεί τότε έχει λυθεί και το πρόβλημα. Στο σχήμα 5.17 βλέπουμε τον γράφο, όπου διασχίζει την ιεραρχία η συνάρτηση Execute.

```
function Execute(level e, state s, action a)
    if (e = 0)
        execute primitive action a and observe s and reward
        return
    m ← sub-MDP associated with action a
    repeat
        act ← value( level e, sub-MDP m, state s) or exploration policy
        ls^e ← abstraction of s at level e
        Execute(level e − 1, state s, action act)
        if((s, act)=exit) return
        else
            V' ← value( level e, sub-MDP m, state s)
            E_m(ls^e, act) = (1 − β)E_m(ls^e, act) + β(reward + V')
end Execute
```

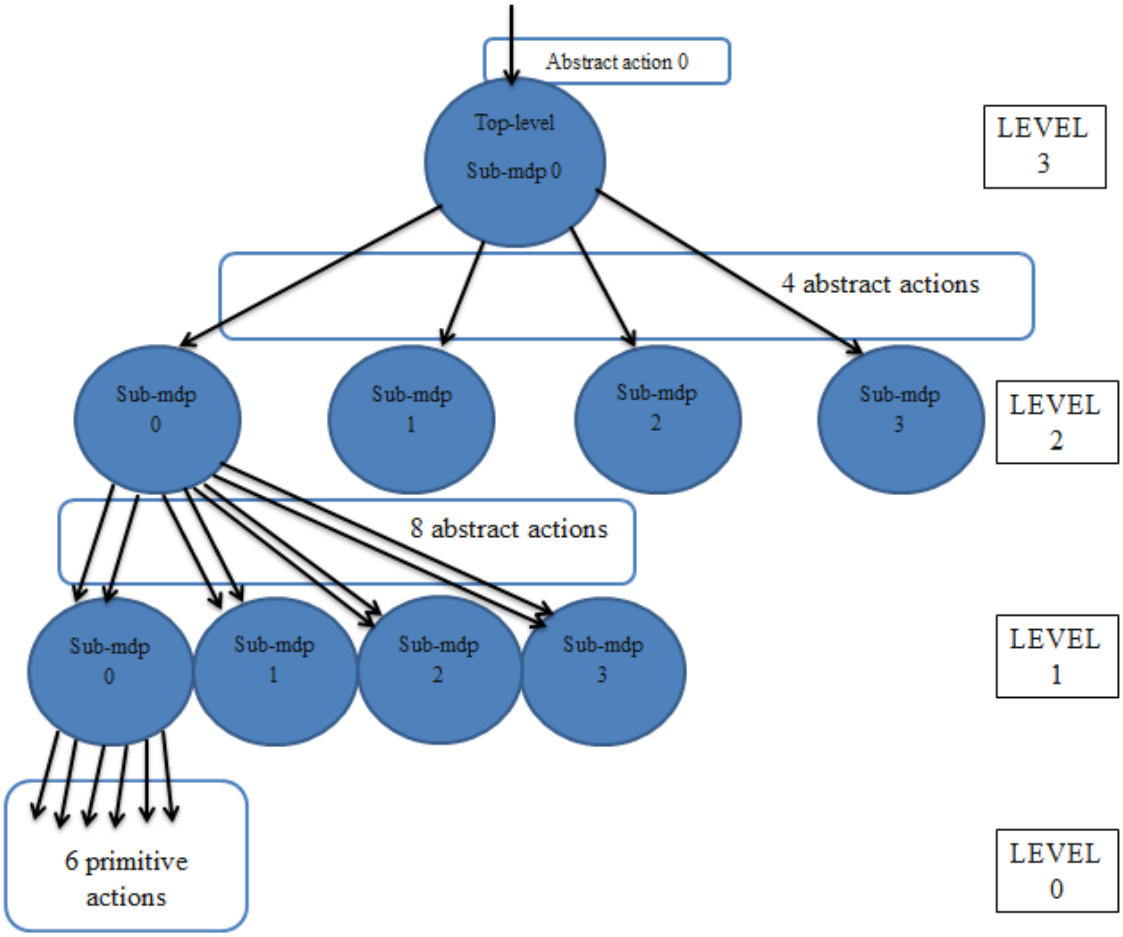
Σχήμα 5.15: Ο αλγόριθμος της συνάρτησης Execute που λύνει το πρόβλημα. Η συνάρτηση καλεί αναδρομικά τον εαυτό της με πολιτικές κατώτερων επιπέδων. Η ενέργεια που επιλέγεται κάθε φορά να εκτελεστεί επιλέγεται μέσω της συνάρτησης value (σχήμα 5.16) ή κάποιας άλλης πολιτικής (υλοποιήθηκε η e-greedy πολιτική σαν $2^{η}$ επιλογή). Η τιμή ls^{e} αποτελεί την τιμή της μεταβλητής του τρέχοντος επιπέδου (που εξ' αρχής δόθηκε με τυχαίο τρόπο απ' το πρόγραμμα). Δηλαδή έστω ότι το πρόγραμμα όρισε : taxi=x, passenger=y, destination=z, το ls^{e}=x για e=1, ls^{e}=y για e=2, ls^{e}=z για e=3. Επίσης η τιμή V' που χρησιμοποιείται στην εξίσωση E κάθε φορά επιλέγεται μέσω της συνάρτησης value (σχήμα 5.16). Ο αλγόριθμος έχει παρθεί απ' τον Hengst (2002).

```
function value( level e; sub-MDP m; hierarchical state s )
    if e = 0 return 0
    A ← actions available in m
    v ← −∞
    a^{e*} ← undefined (best greedy action at level e)
    for each a ∈ A
        m_{e−1} ← sub-MDP associated with a at level e − 1
        v' ← value(e − 1, m_{e−1}, s)+E*_m(s^e, a)
        if (v' > v)
            v ← v'
            a^{e*} ← a
    return v, a^{e*}
end value
```

Σχήμα 5.16: Ο αλγόριθμος της συνάρτησης value βρίσκει τη βέλτιστη ενέργεια για το sub-MDP m του επιπέδου e και τη βέλτιστη τιμή v βάση της κατάστασης s του προβλήματος. Η συνάρτηση καλεί αναδρομικά τον εαυτό της με το sub-MDP του κατώτερου επιπέδου για να διασχίσει όλη την ιεραρχία του δέντρου και να βρει τις βέλτιστες τιμές. Ο αλγόριθμος έχει παρθεί απ' τον Hengst (2002). Υλοποιήθηκε η ομώνυμη συνάρτηση *Value(int e, sub_mdp m, State_Variables s) (δες ΠΑΡΑΡΤΗΜΑ Α, σελ. 35).*

5.2.8 Γράφος HexQ

Είναι το δέντρο το οποίο δημιουργεί ιεραρχικά ο αλγόριθμος HexQ και στο οποίο γινόταν αναφορά και σε προηγούμενα κεφάλαια (σχήμα 5.17). Ένας άκυκλος, κατευθυνόμενος γράφος, όπου μέσω αυτού γίνεται ιεραρχική διάσχιση μέσω της Execute (σχήμα 5.15)με στόχο τη λύση του ολικού ΜΔΑ. Έχει βάθος d-1, όπου d ο αριθμός των μεταβλητών του προβλήματος. Αποτελείται από κόμβους οι οποίοι αντικατοπτρίζουν τα sub-MDPs που δημιουργήθηκαν σε κάθε επίπεδο. Ενώ τα βέλη αντικατοπτρίζουν τις διαθέσιμες primitive ή abstract ενέργειες που μπορεί να επιλεγούν σε κάθε sub-MDP.



Σχήμα 5.17: Η συμπαγής αναπαράσταση του HexQ γράφου αφού έχει γίνει διάσπαση του ολικού ΜΔΑ σε υπο-ΜΔΑ. Το κάθε sub-MDP έχει στη διάθεση του ένα αριθμό ενεργειών, 6 για τα sub-MDPs του $1^{ου}$ επιπέδου, 8 για τα sub-MDPs του $2^{ου}$ επιπέδου και 4 για τα sub-MDPs του $3^{ου}$ επιπέδου. Το κάθε sub-MDP τερματίζεται μέσω της μοναδικής εξόδου του και η αναδρομή επιστρέφει στο προηγούμενο επίπεδου (το ανώτερο του). Όταν η αναδρομή φτάσει στο επίπεδο 3, στον μοναδικό κόμβο του τελευταίου επιπέδου, τότε το πρόβλημα τερματίζεται. Ο γράφος αυτός βρίσκεται στο *ArrayList Sub-Mdps και οι αφαιρετικές ενέργειες των δυο επιπέδων στο ArrayList Abstract_Actions* ενώ οι primitive ενέργειες στο *ArrayList Primitive_Actions.*

Κεφάλαιο 6

Υλοποίηση HexQ

Σ’ αυτό το κεφάλαιο καταγράφηκαν αναλυτικά όλες οι υποθέσεις, προσεγγίσεις και τεχνικές που χρησιμοποιήθηκαν ούτος ώστε να αντιμετωπιστούν τα προβλήματα που παρουσιάστηκαν κατά το τέλος της υλοποίησης. Εν τέλει δεν λύθηκαν αυτά τα προβλήματα, αυτό πιστεύω οφείλεται στους παράγοντες που αναφέρω στο κεφάλαιο αυτό λεπτομερώς.

6.1 Πορεία εργασίας

Τα προβλήματα αφορούν το τελευταίο μέρος του αλγόριθμου HexQ, πιο συγκεκριμένα εμπλέκονται σ’ αυτό οι συναρτήσεις ValueIteration() (σχήμα 2.2), Value() (σχήμα 5.16) και Execute() (σχήμα 5.15). Οι συναρτήσεις αυτές ήταν διαθέσιμες σε μορφή ψευδοκώδικα στη διδακτορική εργασία του Hengst (2002), υλοποιήθηκαν έγκαιρα αλλά το πρόγραμμα δεν επέστρεφε ορθά αποτελέσματα για διάφορους λόγους. Ο πράκτορας ουσιαστικά, εκ των αποτελεσμάτων, δεν μάθαινε με βέλτιστο τρόπο και σε βέλτιστο χρόνο να μεταφέρει τον επιβάτη στον προορισμό του.

Στην προσπάθεια εύρεσης του προβλήματος και αφού έγινε η απαραίτητη διαδικασία αποσφαλμάτωσης του κώδικα επανειλημμένα, οδηγήθηκα σε πολλά

αδιέξοδα στο τελευταίο μέρος του. Ο διδακτορικός φοιτητής Β.Βασιλειάδης με βοήθησε ιδιαίτερα σ' αυτό το στάδιο, λόγω των γνώσεων και της εμπειρίας που κατέχει στον τομέα της ΕΜ, με στόχο την εύρεση του προβλήματος που παρουσιάστηκε. Στην διδακτορική εργασία του Hengst (2002), όπου ήταν η βασική πηγή πληροφοριών μου, δεν υπήρχαν οι απαραίτητες επεξηγήσεις (στη λεπτομέρεια που πιστεύω ότι θα έπρεπε να υπήρχαν) για τα παραρτήματα ψευδοκώδικα των τριων συναρτήσεων που ανέφερα πιο πάνω. Υπήρξαν πολλές ασάφειες εξ' αρχής, οι οποίες αρχικά δεν λήφθηκαν υπόψη, διότι παρόλη την δυσκολία που αντιμετώπισα στην κατανόηση αλλά και στο προγραμματιστικό μέρος της εργασίας, το προγραμματιστικό μέρος προχωρούσε κανονικά εντός των χρονοδιαγραμμάτων. Επίσης κάποια σημεία της εργασίας δεν διασαφηνίζονταν όσο έπρεπε, οπόταν δεν μπορούσα να βοηθηθώ ιδιαίτερα απ' το θεωρητικό μέρος και σε αυτές τις περιπτώσεις χρειάστηκε να γίνουν κάποιες υποθέσεις.

Αρχικά ο αλγόριθμος HexQ προϋποθέτει μια τυχαία εξερεύνηση με στόχο να συλλέξει 1) στατιστικά στοιχεία (για στοχαστικές μεταβάσεις στο πρόβλημα στα πλαίσια ενός στοχαστικού περιβάλλοντος), 2) την συχνότητα αλλαγής των μεταβλητών του προβλήματος (χρησιμοποιείται στα αρχικό στάδιο του αλγορίθμου) 3) να μάθει τη συνάρτηση μεταβάσεων και 4) τη συνάρτηση αμοιβής του προβλήματος. Αυτό ήταν αδύνατο στα αρχικά στάδια της υλοποίησης διότι για να γίνει η τυχαία εξερεύνηση στο περιβάλλον του προβλήματος και να συλλεχθούν όλες οι απαραίτητες πληροφορίες, έπρεπε να υλοποιηθεί εξ' ολοκλήρου ο αλγόριθμος HexQ, οπόταν η διαδικασία αυτή αναστάληκε για το τέλος της υλοποίησης (η διαδικασία αυτή αναφέρεται στην $1^{η}$ παράγραφο της σελίδας 106 της διδακτορικής εργασίας). Τα στατιστικά στοιχεία δεν αφορούσαν το TP, αφού δεν αφορά στοχαστικό περιβάλλον και δεν ήταν στα πλαίσια της

εργασίας μου, οπόταν μπορούσαν να παραληφθούν. Επίσης το μοντέλο θεωρήθηκε αρχικά ότι ήταν γνωστό διότι δεν διασαφηνιζόταν πώς στα πλαίσια αυτής της τυχαίας εξερεύνησης ο πράκτορας θα ανακάλυπτε το μοντέλο χωρίς καμία γνώση εκ των προτέρων, χωρίς τη χρήση των δεδομένων του προβλήματος. Έτσι δημιούργησα τις δυο αυτές συναρτήσεις που «μαθαίνουν» το μοντέλο με βάση τα δεδομένα που είχα στη διάθεση μου. Έλαβα επίσης ως δεδομένο τη συχνότητα αλλαγής των μεταβλητών του προβλήματος, που ήταν μια μικρή λεπτομέρεια που θα μεταφερόταν στο τέλος, αφού στα αρχικά στάδια ήταν αδύνατο να γίνει τυχαία εξερεύνηση στο σύνολο καταστάσεων του προβλήματος για να βρεθεί η συχνότητα αλλαγής των μεταβλητών.

Περί τα μέσα Ιανουαρίου είχαμε ενδοιασμούς για το αν έπρεπε να υλοποιηθεί η συνάρτηση ValueIteration(). Λόγω της πολυπλοκότητας της εργασίας αυτής, ήταν χρονοβόρα τόσο η διαδικασία κατανόησης του προβλήματος σε βάθος, όσο και η διαδικασία υλοποίησης του κάθε μέρους του αλγορίθμου ξεχωριστά, κι έτσι σκεφτήκαμε να χρησιμοποιήσουμε τον αλγόριθμο Q-Learning αντί του ValueIteration(), με στόχο την εξοικονόμηση χρόνου (λόγω της απλότητας του $1^{ου}$). Τελικά δεν έγινε αυτό, διότι πρώτον, ο Q-Learning πιθανόν δεν θα επέφερε τα ίδια αποτελέσματα με τη συνάρτηση ValueIteration() και δεύτερον θα ήθελε περισσότερη ώρα να καταλήξει σε αποτελέσματα ο Q-Learning, κάτι που δεν θα ίσχυε με την ValueIteration().

Η συνάρτηση ValueIteration() ουσιαστικά είναι η συνάρτηση η οποία λύνει τα sub-MDPs, βρίσκει την βέλτιστη συνάρτηση E, δηλαδή την αναμενόμενη τιμή των μελλοντικών αμοιβών απ’ όλες τις καταστάσεις, με όλες τις ενέργειες. Το

πρόβλημα εκ πρώτης όψεως θα λυνόταν πολύ πιο εύκολα διότι θα μάθαινε ο πράκτορας από εκείνο το σημείο και τι καταστάσεις και ενέργειες με τις καλύτερες αξίες και εκτελώντας την Execute θα τερματιζόταν το πρόβλημα (με την λύση του ανώτατου sub-MDP). Όμως επειδή στον Hengst (2002) αναφέρεται το εξής: «The sub-MDPs are solved using value iteration with the transition model (transition probabilities and rewards) determined from the recorded statistics in line 5 of the HEXQ algorithm» (σελίδα 107) και εφόσον αυτή η τυχαία εξερεύνηση όπως εξήγησα πιο πάνω ήταν αδύνατο να γίνει (και θεωρήσαμε ότι ξέρουμε το μοντέλο), ο πράκτορας είτε με την χρήση είτε όχι, της συνάρτησης ValueIteration() θα μάθαινε τη βέλτιστη διαδρομή ούτως η άλλως με τη συνάρτηση Execute() στο τελευταίο στάδιο και γι' αυτό το λόγο αποφασίσαμε να την αγνοήσουμε για τη δεδομένη στιγμή.

Στη συνέχεια υλοποιήθηκε εξ' ολοκλήρου το πρόγραμμα γύρω στα τέλη Μαρτίου και τότε χρειάστηκε να αναθεωρηθούν κάποιες λεπτομέρειες οι οποίες ήταν καθοριστικές για την επίδοση του αλγορίθμου, αφού τα πρώτα αποτελέσματα (χωρίς την χρήση της ValueIteration()) υποδείκνυαν ότι ο πράκτορας δεν μάθαινε (παρόλο που αναφέρεται ότι στο τελευταίο επίπεδο με τη συνάρτηση Execute() επίσης μαθαίνει τη βέλτιστη διαδρομή. Στις επόμενες παραγράφους ακολουθούν όλες οι δοκιμές που έγιναν για να λυθεί το πρόβλημα.

Σχήμα 6.1: Ο ψευδοκώδικας που φαίνεται στο δείχνει με αφαιρετικό τρόπο την ροή του προγράμματος από την κλάση Trials *(ΠΑΡΑΡΤΗΜΑ Α, σελ. 44)* στην κλάση HexQ *(ΠΑΡΑΡΤΗΜΑ Α, σελ. 1)*.

Μια $1^{η}$ δοκιμή ήταν ή αρχικοποίηση όλων των τιμώς των πινάκων Ε (το κάθε sub-MDP έχει το δικό του πίνακα Ε) με μια μεγάλη τιμή (έστω 10000) και συγχρόνως ενεργοποιήθηκε αποκλειστικά η e-greedy πολιτική με τιμή e=0. Αυτή η μέθοδος ονομάζεται “Optimistic Initialization” και κανονικά έπρεπε να γίνεται καλή εξερεύνηση του περιβάλλοντος. Δυστυχώς τα αποτελέσματα ήταν ακριβώς τα ίδια.

Επίσης, εφαρμόστηκαν τεχνικές “Tie breaking” στη συνάρτηση Value(σχήμα 5.16), διότι σε περίπτωση που οι αξίες v ήταν ισοδύναμες για περισσότερες από μία ενέργειες, να μην λαμβάνε΄ται πάντα η $1^{η}$ ενέργεια ως αυτή που δίνει την μέγιστη τιμή v, αλλά να επιλέγεται μια ενέργεια τυχαία από ένα σύνολο με τις ισοδύναμες αξίες.

Στη συνέχεια η προσοχή στράφηκε προς τη συνάρτηση Execute() και πιο συγκεκριμένα στην τερματική συνθήκη της συνάρτησης, όπου τερματίζεται

συγχρόνως και το ολικό πρόβλημα στο τελευταίο επίπεδο της αναδρομής. Στον ψευδοκώδικα της συνάρτησης αυτής, στη γραμμή 9 (σχήμα 5.15), αναγράφεται η συνθήκη: if ((s, act)=exit) return. Αυτό ήταν κάτι πολύ σχετικό, χωρίς ιδιαίτερη διευκρίνιση, οπόταν σκεφτήκαμε μήπως οι τερματικές συνθήκες που αρχικά θεωρήσαμε, ήταν λανθασμένες. Θεωρήθηκε ότι με οποιαδήποτε έξοδο του επιπέδου θα μπορούσε να κάνει return στη γραμμή εκείνη, αλλά ουσιαστικά έπρεπε να κάνει return μόνο από την μοναδική έξοδο του τρέχοντος sub-MDP m. Αυτό διορθώθηκε άμεσα αλλά και πάλι δεν ήταν αυτό το πρόβλημα.

Στη συνέχεια αναθεωρήθηκε η συνάρτησης αμοιβής R. Ελέγχθηκε και επιβεβαιώθηκε ότι πριν την ορθή έξοδο από το τρέχον sub-MDP η αμοιβή είναι ίση με 20 (επιτυχής έξοδος από το sub-MDP m), ενώ για τις υπόλοιπες κινήσεις ήταν αναλόγως -1, -10.

Γύρω στα μέσα Απριλίου παρατηρήσαμε στη σελίδα 94 της διδακτορικής εργασίας του Hengst (2002), ότι αναγράφεται στην υποσημείωση στο τέλος της σελίδας το εξής: “The standard process is only complicated by the need to recursively calculate the value of the post transition states s’ from the decomposed value function…This means that the value function in…becomes V(s’)=value(e, sub-MDP m, s’)”. Οπότε στη συνάρτηση ValueIteration() αντί να βρίσκουμε την ενέργεια a που οδηγεί τον πράκτορα από την κατάσταση s στην κατάσταση s’ με την μέγιστη τιμή V (αξία), θα καλούμε την συνάρτηση value(σχήμα 5.16). Καμία βελτίωση στα αποτελέσματα.

Σε αυτό το σημείο ήταν προφανώς απαραίτητη η υλοποίηση της ValueIteration(). Το πρόβλημα που εμφανίστηκε στην υλοποίηση ήταν το εξής: Στη γραμμή 7 του

ψευδοκώδικα της συνάρτησης, υπάρχει η συνάρτηση αμοιβής R. Αυτή η συνάρτηση επιστρέφει την αμοιβή του πράκτορα για κάθε primitive ενέργεια που εκτελεί. Το πρόβλημα ήταν ότι η συνάρτηση ValueIteration() καλείται από τον αλγόριθμο HexQ και για την λύση των sub-MDPs του 2^{ου} επιπέδου. Στο 2^{ο} επίπεδο οι ενέργειες είναι αφαιρετικές. Δημιουργήθηκε το εξής ερώτημα «Πώς θα δίνονται οι αμοιβές σε αυτό το επίπεδο εφ' όσον οι ενέργειες είναι αφαιρετικές». Αρχικά υλοποιήθηκε για το 1^{ο} επίπεδο μόνο, για εξοικονόμηση χρόνου και παρουσίαζε-ορθά-ατέρμονα βρόγχο και εκεί παρατηρήσαμε κάτι πολύ σημαντικό. Ο πράκτορας δεν μπορούσε να μάθει, παρόλο που οι αμοιβές κατανέμονταν ορθά για κάθε του κίνηση. Τότε σκεφτήκαμε ότι για να μάθει ο πράκτορας μια πιθανή λύση στο πρόβλημα του ατέρμονα βρόγχου είναι στις εξόδους να δίνεται αμοιβή ίση με μηδέν. Αυτό έλυσε το πρόβλημα και μέσα σε λίγες δοκιμές τερμάτιζε η ValueIteration() λύνοντας ορθά τα sub-MDPs του 1^{ου} επιπέδου. Συνοψίζοντας για το 1^{ο} επίπεδο, οι ψευδοαμοιβές είναι οι εξής: 0 για έξοδο από το sub-MDP, για GET ΚΑΙ PUT -10, για οτιδήποτε άλλο -1.

Όσο αφορά το 2^{ο} επίπεδο, τα πράγματα είναι κάπως πιο πολύπλοκα. Όσο αφορά τις αμοιβές, δίνεται και πάλι αμοιβή ίση με μηδέν όταν γίνεται έξοδος από το τρέχον sub-MDP και για τις υπόλοιπες περιπτώσεις κατανέμονται οι αμοιβές αναλόγως:

- Αν γίνει GET ενώ ο επιβάτης δεν είναι στη τοποθεσία που έγινε το GET, αμοιβή ίση με -10.
- Αν γίνει PUT ενώ ο επιβάτης δεν είναι στο ταξί, αμοιβή ίση με -10.
- Αν γίνει PUT στη θέση που ορίζει η επιλεγμένη αφαιρετική ενέργεια, αμοιβή ίση με -1 και αλλάζει η θέση επιβάτη.

- Αν γίνει GET από τη θέση που ορίζει η επιλεγμένη αφαιρετική ενέργεια, αμοιβή ίση με -1 και αλλάζει η θέση επιβάτη.

Οι πιο πάνω υποθέσεις όσο αφορά τις ψευδοαμοιβές (συνάρτηση *simple_TakeAction(int action, int s))* έγιναν διότι ο πράκτορας δεν συνέδεε τις κατάλληλες καταστάσεις με θετικές αμοιβές και γι' αυτό δεν μπορούσε να ανακαλύψει την σωστή πολιτική. Όταν ο επιβάτης-στο επίπεδο 3 (στη συνάρτηση Execute())- μεταφερθεί στον προορισμό του, τότε να δίνεται αμοιβή ίση με 20.

Μια άλλη ασάφεια που πιθανόν να οφείλεται στο πρόβλημα του ατέρμονα βρόγχου στο $2^{ο}$ επίπεδο και αφού διασαφηνίστηκαν οι ψευδοαμοιβές του, είναι μια λεπτομέρεια η οποία διαδραματίζει καθοριστικό ρόλο στη εκτέλεση της ValueIteration() κατά την άποψη μου. Στο $2^{ο}$ επίπεδο καλείται αναδρομικά η συνάρτηση Value() διασχίζοντας το $1^{ο}$ επίπεδο. Στη γραμμή 7 του ψευδοκώδικα της Value(), για εύρεση του $E^*[s^e][a]$, η μεταβλητή s^e δεν γνωρίζουμε ξεκάθαρα ποια τιμή αντικατοπτρίζει, δεν ορίζεται ρητά κάπου. Υποθέσαμε ότι είναι η τιμή που πρέπει να πάρει στο τέλος της αφαιρετικής ενέργειας. Το πρόβλημα στην ValueIteration() για το επίπεδο 2 εν τέλει δεν λύθηκε. Δεν γνωρίζουμε αν υπάρχει κάποιο άλλο σημείο το οποίο θα έπρεπε να ενισχύσουμε με υποθέσεις ή αν κάτι παραλείπεται απ' τον ψευδοκώδικα ή στην επεξήγηση. Επίσης κάποια σημεία που δεν διευκρινίζονταν περισσότερο ήταν για παράδειγμα στην σελίδα 94 " It is important to remember to restrict actions for other exit states to non-exit actions during learning, even for exploration, as unintended exits at any level will mislead the learner" αλλά και στην σελίδα 162 "The introduction of non-exiting sub-MDPs that are interrupted during learning and the extension of on-policy pseudo reward

functions with large positive rewards on exit has allowed HEXQ to automatically decompose and solve infinite horizon MDPs with positive rewards".

Γενικά, ο πράκτορας με τη συνάρτηση ValueIteration() υποβοηθείται, διότι γνωρίζει τις E τιμές όμως το μόνο που μαθαίνει είναι να εξέρχεται ορθά απ' την μοναδική κατάσταση εξόδου του sub-MDP που βρίσκεται και να τερματίζει την επιλεγμένη αφαιρετική ενέργεια. Και στη συνέχεια στην Execute(), θα έπρεπε κατά τη γνώμη μου, να εμπλέκονται οι μεταβλητές του προβλήματος (οι τυχαίες τιμές που δόθηκαν αρχικά στις μεταβλητές *θέση ταξί, θέση επιβάτη, προορισμός*) κάτι που δεν συμβαίνει στη ValueIteration().

Εν κατακλείδι, αυτός ο αλγόριθμος με βάση τον Hengst (2002) υπερέχει των υπόλοιπων ιεραρχικών αλγορίθμων που χρησιμοποιούνται στην IEM, έτσι ήταν αναμενόμενο να παρήγαγε πολύ καλά αποτελέσματα. Παρόλα αυτά, το πρόβλημα στην ValueIteration() δεν μας επέτρεψε να συνεχίσουμε, αφού δεν γνωρίζαμε τι άλλες πιθανές υποθέσεις θα μπορούσαμε να κάνουμε. Ήταν μια πολύμηνη υλοποίηση (η οποία άρχισε στα τέλη Δεκεμβρίου και τερματίστηκε στις αρχές Απριλίου χωρίς καμία διακοπή). Πιστεύω ότι το δύσκολο μέρος της εργασίας μου ήταν να κατανοήσω κάθε λεπτομέρεια του αλγορίθμου (λόγω της πολύπλοκης φύσης του), αλλά, όπως προανέφερα επανειλημμένα λόγω της ύπαρξης αρκετών αδιευκρίνιστων σημείων στο τελικό στάδιο επιβράδυνε την διαδικασία υλοποίησης.

6.2 Συστατικά του Προγράμματος

Το πρόγραμμα αποτελείται από 11 κλάσεις τις οποίες δημιούργησε, 2 κλάσεις που χρησιμοποιώ από τον Λάμπρου (2011) ενώ στο πρόγραμμα βρίσκονται και άλλες κλάσεις του Λάμπρου (2011) τις οποίες όμως δεν χρησιμοποίησα στην υλοποίηση μου.

Η βασική κλάση είναι η HexQ που αντικατοπτρίζει τον ομώνυμο αλγόριθμο HexQ, περιέχει 43 συναρτήσεις και καλείται από την κλάση Trials *(ΠΑΡΑΡΤΗΜΑ Α, σελ. 44)* η οποία είναι κλάση καθαρά διαδικαστική, ελέγχει την ροή του προγράμματος και αποτελείται από 6 συναρτήσεις. Οι υπόλοιπες κλάσεις μου, αποτελούν κατασκευαστές. Οι δυο κλάσεις που δανείστηκα απ' τον Λάμπρου (2011) είναι η State_Variables *(ΠΑΡΑΡΤΗΜΑ Α, σελ. 53)* και η SA_Gridworld *(ΠΑΡΑΡΤΗΜΑ Α, σελ. 47)*.

Όλες οι κλάσεις οι οποίες προανέφερα (αυτές που δημιούργησα και οι δυο που δανείστηκα απ' τον Λάμπρου (2011)) βρίσκονται στο Παράρτημα Α ενώ οι κλάσεις οι οποίες υπάρχουν στο πρόγραμμα ενώ δεν χρησιμοποιήθηκαν δεν εμφανίζονται στο Παράρτημα Α.

Κεφάλαιο 7

Συμπεράσματα και Μελλοντική Εργασία

Παρόλο που στα πλαίσια της εργασίας μου δεν καταλήξαμε σε αποτελέσματα ούτως ώστε να είμαστε σε θέση να αντικρούσουμε ή να επιβεβαιώσουμε τις απόψεις του Hengst για την απόδοση του αλγόριθμου HexQ, είμαστε σε θέση να δηλώσουμε ότι είναι ένας αλγόριθμος που υπερτερεί των υπολοίπων αλγόριθμων που ανήκουν στην κατηγορία των ιεραρχικών μεθόδων, γενικότερα υπερέχει των αλγόριθμων που χρησιμοποιούνται για να λύσουν προβλήματα IEM. Αυτό είναι κάτι το αναμφισβήτητο, διότι η δομή και λειτουργία του αλγόριθμου η οποία αποσυνθέτει αυτόματα και χειρίζεται προβλήματα IEM, είναι πρωτοφανής σε σχέση με χειρισμούς άλλων αλγόριθμων, οι οποίοι δεν χειρίζονται με παρόμοιο τα προβλήματα αυτά. Αυτό δεν είναι κατ' ανάγκην κακό, απλά ο HexQ με τη συμπίεση πολυδιάστατων μαρκοβιανών διαδικασιών απόφασης, την αυτοματοποίηση της διαδικασίας διάσπασης, τον αφαιρετικό τρόπο απλοποίησης των καταστάσεων (state abstraction) του προβλήματος, την εφαρμογή του σε στοχαστικά και μη περιβάλλοντα και την ευρέως διαδεδομένη χρήση του σε πολλές εφαρμογές όπως π.χ. Οι πύργοι του Hanoi, τα 25 δωμάτια, το πρόβλημα του ταξί (και οι παραλλαγές του) κλπ, του δίνουν ένα προβάδισμα σε σχέση με τους υπόλοιπους αλγόριθμους που πιθανόν να λύνουν τα ίδια προβλήματα.

Αναμφισβήτητα, αξίζει να ασχοληθεί κάποιος με τον αλγόριθμο αυτό, έναν πολλά υποσχόμενο αλγόριθμο, ο οποίος πιστεύω ότι όντως είναι σε θέση να λύσει και να χειριστεί διαφορετικά είδη προβλημάτων με βέλτιστο τρόπο. Η παρούσα υλοποίηση είναι πιστεύω πολύ κοντά στα αποτελέσματα οπόταν εάν κάποιος ασχοληθεί με την δική μου εργασία και πιστεύω ότι μπορεί όντως να θα οδηγηθεί στα πολυαναμενόμενα αποτελέσματα που καταγράφονται στην εργασία του Hengst (2002). Ούτως η άλλως με την χρήση της δικής μου εργασίας ή όχι, κατά την άποψη μου (και μετά από πολύμηνη μελέτη), πιστεύω πως είναι ένας αλγόριθμος που αξίζει και πρέπει να μελετηθεί διότι διαθέτει όλες τις προοπτικές για βελτιστοποίηση, ενίσχυση της απόδοσης πολλών προβλημάτων IEM και όχι μόνο.

Αναφορές

Bowling, M., & Veloso, M. (2002). Multiagent learning using a variable learning rate. *Artificial Intelligence*, 136(2), 215-250.

Dietterich, T. G. (2000). Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research, 13*, 227-303.

Diuk, C., Cohen, A., & Littman, M. L. (2008). An object-oriented representation for efficient reinforcement learning. In Proceedings of the 25th international conference on Machine learning, ACM, New York, NY, 240-247.

Hengst, B. (2002). Discovering Hierarchy in Reinforcement Learning with HEXQ. In: Sammut, C., Hoffmann, A. (eds), Proceedings of the Nineteenth International Conference on Machine Learning (ICML), Sydney Australia, Morgan-Kaufman, 243–250.

Hengst, B. (2003). Discovering Hierarchy in Reinforcement Learning, PhD Thesis, Computer Science and Engineering, University of New South Wales, Sydney Australia.

Hengst, B. (2004). Model approximation for hexq hierarchical reinforcement learning. In Machine Learning: J.-F. Boulicaut *et al.* (eds) : 15th European Conference on Machine Learning (ECML), Pisa, Italy, Springer-Verlag, LNAI 3201, 144-155.

Hengst, B. (2011), "Hierarchical Approaches", in Wiering, M. & van Otterlo, M. (eds), Reinforcement Learning: State of the Art (Vol.12), Springer series Adaptation, Learning, and Optimization, Springer, 293-323.

Kaelbling, L.P., Littman, M.L. & Moore, A.W. (1996), Reinforcement Learning: A Survey, *Journal of Artificial Intelligence Research, 4*, 237-285.

Lambrou, I., Vassiliades, V. and Christodoulou, C. (2012). An extension of a hierarchical reinforcement learning algorithm for multiagent settings. Recent Advances in Reinforcement Learning, EWRL (European Workshop on Reinforcement Learning) 2011, Lecture Notes in Artificial Intelligence (Subseries of Lecture Notes in Computer Science), eds. S. Sanner and M. Hutter, 7188, Berlin: Springer-Verlag, 261-272.

Rummery, G. A. & Niranjan, M. (1994), On-line Q-learning using connectionist systems. Technical Report CUED/F-1NFENG/TR 166, Cambridge University Engineering Department.

Watkins, C. J. C. H. (1989). Learning from delayed rewards (Doctoral dissertation, University of Cambridge).

Λάμπρου, Ι. (2011) Ιεραρχική Μάθηση σε προβλήματα ενός πράκτορα και πολλαπλών πρακτόρων. Ατομική Διπλωματική Εργασία, Τμήμα Πληροφορικής, Πανεπιστήμιο Κύπρου.

ΠΑΡΑΡΤΗΜΑ Α

ΠΗΓΑΙΟΣ ΚΩΔΙΚΑΣ

```
/**
 * Klasi i opoia ilopoiei ton algorithmo HexQ gia to Taxi Problem
 *
 * @author Maria Kalli
 *
 */
import java.io.FileNotFoundException;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.Random;

public class HexQ  {
	public static ArrayList<ArrayList<Integer>> X = new
ArrayList<ArrayList<Integer>>();// periexei
                                                                  // ta 3 state
                                                                  // variables
tou TP
	public static ArrayList<ArrayList<Integer>> States = new
ArrayList<ArrayList<Integer>>();////olwn twn levels
	public static ArrayList<Integer> Entries = new
ArrayList<Integer>();//tou current level
	public static ArrayList<ArrayList<ExitNode>> Exits = new
ArrayList<ArrayList<ExitNode>>();//olwn twn levels
	public static ArrayList<ArrayList<MER>> MERS = new
ArrayList<ArrayList<MER>>();//olwn twn levels
	public static ArrayList<ArrayList<ExitNode>> Abstract_Actions = new
ArrayList<ArrayList<ExitNode>>();// Action(s,a) twn 2 levels
	public static ArrayList<Integer> Primitive_Actions = new
ArrayList<Integer>();//tou 1ou epipedou
	public static ArrayList<ArrayList<graph_node>> Graph = new
ArrayList<ArrayList<graph_node>>();// periexei


	// tous


	// grafous


	// twn


	// levels
	public static ArrayList<scc_couples> Connected_sccs = new
ArrayList<scc_couples>();// couples
```

```
                                                                    // of
                                                                    // nodes
                                                                    // connecting
                                                                    // 2
                                                                    // different
                                                                    // sccs
      public static ArrayList<ArrayList<sub_mdp>> Sub_Mdps = new
ArrayList<ArrayList<sub_mdp>>();// periexei


      // ta


      // sub_mdps


      // twn 3


      // levels
      public static ArrayList<Integer> Actions = new
ArrayList<Integer>();//integer representation
       static PrintWriter wr1;
      public static Random r = new Random();
      public static int trials;
      public static int episodes;
      public static String filename;
      public static int[] episodeSteps;//steps twn epeisodiwn enos trial
      public static int[] episodeRewards;//rewards twn epeisodiwn enos
trial

      //private double[] timePerEpisode;
      private static int steps = 0;
      private static int sum_rewards = 0;
      private static int reward = 0;
      private static State_Variables state;
      private static double epsilon;
      private static double learning_rate;// 0.25
      private static double decay;
      private static double discount_factor;//1
      private static int depth;
      private static int finTime;
      private static int SCClabel;
      private static char[] col;
      private static int[] f;
      private static int[] SCC;// label tou SCC
      private static int[][] adj;


      //private static double v;
```

```
	// six primitive actions encoded
	public final static int N = 0;
	public final static int S = 1;
	public final static int E = 2;
	public final static int W = 3;
	public final static int GET = 4;
	public final static int PUT = 5;


/*sinartisi pou epistrefei to va8os tis ierarxias*/
	public static int getDepth() {
		return depth;

	}
	/*sinartisi pou dimiourgei ton pinaka me ta steps twn episodiwn*/
	public static void set_Episode_Steps() {
		episodeSteps = new int[episodes];
	}

	/*sinartisi pou lamvanei to onoma tou arxeiou pou dinei o xristis*/
	public static void insertFilename() {
		System.out.println("Please insert the name of the input
file:");
		filename = StdIn.readString();
	}
	/*sinartisi pou lamvanei to epitrepto learning rate*/
	public static void chooseLearningRate() {

		do {
			System.out
					.println("Please set the learning rate
(between 0 and 1)");
			learning_rate = StdIn.readDouble();
		} while (learning_rate < 0 || learning_rate > 1);

	}
	/*sinartisi pou lamvanei to epitrepto discount factor*/
	public static void chooseDiscFactor() {

		do {
			System.out.println("Please set the discount_factor
(between 0 and 1)");
			discount_factor = StdIn.readDouble();
		} while (discount_factor < 0 || discount_factor > 1);

	}
	/*sinartisi pou lamvanei to epitrepto decay*/
	public static void chooseDecay() {

		do {
			System.out.println("Please set the decay (between 0 and
1)");
			decay = StdIn.readDouble();
		} while (decay < 0 || decay > 1);
```

```
	}
	/*sinartisi pou lamvanei ta trials pou tha ekteleitai o HEXQ*/
	public static void chooseTrials() {
		System.out
				.println("Please enter how many trials you would
like to run:");
		trials = StdIn.readInt();
	}
	/*sinartisi pou lamvanei ta epeisodia*/
	public static void chooseEpisodes() {

		System.out
				.println("Please enter how many episodes you
would like to run:");
		episodes = StdIn.readInt();
	}
	/*sinartisi pou lamvanei to epitrepto epsilon*/
	public static void chooseEpsilon() {

		do {
			System.out.println("Please set epsilon(between 0 and
1)");
			epsilon = StdIn.readDouble();
		} while (epsilon < 0 || epsilon > 1);

	}
	/*sinartisi pou dimiourgei ton pinaka gia ton meso ari8mo vimatwn
ana episodio*/
	public static void set_Avg_Steps_Matrix() {
		Trials.AverageStepsPerEpisode = new double[episodes];
	}
	/*sinartisi pou epistrefei ton ari8mo twn states me vasi to level*/
	public static int getStates(int e) {
		if (e == 1)
			return States.get(e - 1).size();
		else if (e == 2 || e == 3)
			return States.get(e - 1).size() * MERS.get(e -
2).size();
		return 0;
	}
	/*sinartisi pou epistrefei ton ari8mo twn actions me vasi to
level*/
	public static int getActions(int e) {
		if (e == 1)
			return Primitive_Actions.size();
		else if (e == 2 || e == 3)
			return Abstract_Actions.get(e - 2).size();

		return 0;

	}
	/*sinartisi pou epistrefei ton midenismeno pinaka q--> kaleitai gia
arxikopoiisi twn pinakwn olwn twn sub-mdps*/
	public static double[][] initializeQ(int states, int actions) {

		double[][] q = new double[states][actions];
```

```
			for (int s = 0; s < states; s++)
				for (int a = 0; a < actions; a++)
					q[s][a] = 0;
			return q;
		}
		/*sinartisi pou orizei to va8os tis ierarxias tou TP*/
		public static void setDepth() {
			depth = X.size();

		}
		/*sinartisi pou vriskei to transition model tou level 1&2*/
		public static void findTransitionModel(int sizex, int sizey,
				ArrayList<Wall> walls, int e) {

			int[] goals = new int[SA_Gridworld.numOfGoals];
			for (int i = 0; i < SA_Gridworld.numOfGoals; i++)
				goals[i] = SA_Gridworld.getDestLoc(i);
			if (e == 1) {
				ArrayList<graph_node> gr = new ArrayList<graph_node>();
				boolean flag = false;
				boolean flag1 = false;
				boolean flag2 = false;
				boolean flag3 = false;
				boolean flag4 = false;
				int right = -10, left = -10, up = -10, down = -10;

				for (int i = 0; i < (X.get(e - 1).size()); i++) {
					flag = false;
					flag1 = false;
					flag2 = false;
					flag3 = false;
					flag4 = false;

					for (int k = 0; k < SA_Gridworld.wall.size();
k++) {
						Wall temp = SA_Gridworld.wall.get(k);
						//elegxos gia toixous gia na vroume
right,left,up,down apo ka8e katastasi
						if (((temp.cell1 == i) && (temp.cell2 == i
+ 1))
								|| ((temp.cell1 == i + 1) &&
(temp.cell2 == i))) {
							right = i;// wall at right
							flag1 = true;
							// System.out.print(" flag1 enabled
");
						}
						if (((temp.cell1 == i) && (temp.cell2 == i
- 1))
								|| ((temp.cell1 == i - 1) &&
(temp.cell2 == i))) {
							left = i;// wall at left
							flag2 = true;
							// System.out.print(" flag2 enabled
");
						}
```

```
                        if (((temp.cell1 == i) && (temp.cell2 == i
                                    - SA_Gridworld.size_x))
                                    || ((temp.cell1 == i -
SA_Gridworld.size_x) && (temp.cell2 == i))) {
                              up = i;// wall above
                              flag3 = true;
                              // System.out.print(" flag3 enabled
");
                        }
                        if (((temp.cell1 == i) && (temp.cell2 == i
                                    + SA_Gridworld.size_x))
                                    || ((temp.cell1 == i +
SA_Gridworld.size_x) && (temp.cell2 == i))) {
                              down = i;// wall below
                              flag4 = true;
                              // System.out.print(" flag4 enabled
");
                        }
                  }

                  if ((i + 1 < SA_Gridworld.size_x *
SA_Gridworld.size_y)
                              && (((i + 1) % SA_Gridworld.size_x)
!= 0)
                              && (flag1 == false)) {
                        right = i + 1;
                        // System.out.print(" if1 enabled ");
                  } else {
                        right = i;
                        // System.out.print(" else1 enabled ");
                  }
                  if ((i - 1 >= 0) && (i % SA_Gridworld.size_x !=
0)
                              && flag2 == false) {
                        left = i - 1;
                  }
                  // System.out.print(" if2 enabled ");
                  else {
                        left = i;
                        // System.out.print(" else2 enabled ");
                  }
                  if (((i - SA_Gridworld.size_x) >= 0) && (flag3 ==
false)) {
                        up = i - SA_Gridworld.size_x;
                        // System.out.print(" if3 enabled ");
                  } else {
                        up = i;
                        // System.out.print(" else3 enabled ");
                  }
                  if (i + SA_Gridworld.size_x <
(SA_Gridworld.size_x * SA_Gridworld.size_y)
                              && (flag4 == false)) {
                        down = i + SA_Gridworld.size_x;
                        // System.out.print(" if4 enabled ");
                  } else {
                        down = i;
```

```
                              // System.out.print(" else4 enabled ");
                        }

                        for (int a = 0; a < SA_Gridworld.numOfGoals; a++)
{
                              if (i == goals[a]) {
                                    gr.add(new graph_node(i, up, down,
right, left, -100,
                                                -100, i));// e3odos -> to
-100
                                    flag = true;
                                    break;
                              }
                        }
                        if (flag == false)
                              gr.add(new graph_node(i, up, down, right,
left, i, i, i));
                  }
                  graph_node temp;
                  System.out.println();
                  // System.out.println("Transition model for e=1");
                  Graph.add(e - 1, gr);//transition model tou e=1 sti
8esi 0 tou Graph
                  /*
                   * for (int k = 0; k < Graph.get(e-1).size(); k++) {
temp =
                   * Graph.get(e-1).get(k); System.out.println(" node= "
+ k + " up= "
                   * + temp.up + " down= " + temp.down + " right= " +
temp.right +
                   * " left= " + temp.left + " get " + temp.get + " put "
+ temp.put +
                   * " index=" + temp.index); }
                   * System.out.println("gr"+Graph.get(e-1).size());
                   */
            } else {
                  ArrayList<graph_node> gr2 = new
ArrayList<graph_node>();
                  int temp;
                  graph_node temp2;
                  for (int i = 0; i < X.get(e - 1).size(); i++) {
                        temp = X.get(e - 1).get(i);
                        if (temp != -1)
                              gr2.add(new graph_node(X.get(e - 1).get(i),
X.get(e - 1)
                                          .get(i), X.get(e - 1).get(i),
X.get(e - 1).get(i),
                                          X.get(e - 1).get(i), -1,
X.get(e - 1).get(i), i));
                        else
                              // (node 0 only putdown)
                              gr2.add(new graph_node(X.get(e - 1).get(i),
X.get(e - 1)
                                          .get(i), X.get(e - 1).get(i),
X.get(e - 1).get(i),
```

```
                                                X.get(e - 1).get(i), X.get(e -
1).get(i), -100, i));// exit

                                                      // -100

                  }//
                        // System.out.println();
                        // System.out.println("Transition model for
e=2");
                  Graph.add(e - 1, gr2);//transition model tou e=2 sti
8esi 1 tou Graph

                  /*  for (int i = 0; i < Graph.get(e-1).size(); i++) {
temp2 =
                    Graph.get(e-1).get(i); System.out.println(" node= " +
temp2.id +
                    " up= " + temp2.up + " down= " + temp2.down + "
right= " +
                    temp2.right + " left= " + temp2.left + " get=" +
temp2.get +
                    " put=" + temp2.put + " index=" + temp2.index);

                    } System.out.println("gr2="+Graph.get(e-1).size());*/

            }
            // System.out.println();
      }
      /*sinartisi pou vrikskei tis e3odous tou ka8e epipedou*/
      public static void findExits(int e) {
            int[] goals = new int[SA_Gridworld.numOfGoals];

            graph_node temp;
            ExitNode temp2;
            for (int i = 0; i < SA_Gridworld.numOfGoals; i++)
                  goals[i] = SA_Gridworld.getDestLoc(i);
            if (e == 1) {
                  ArrayList<ExitNode> Exits_temp = new
ArrayList<ExitNode>();// (s,a)
                  for (int j = 0; j < Graph.get(e - 1).size(); j++) {
                        for (int i = 0; i < SA_Gridworld.numOfGoals; i++)
                              if (j == goals[i]) {//an einai goal state
                                    temp = Graph.get(e - 1).get(j);
                                    // an prokalei e3odo
                                    if (temp.right == -100)
                                          Exits_temp.add(new
ExitNode(temp.id, new Action(
                                                      temp.id, E)));
                                    if (temp.left == -100)
                                          Exits_temp.add(new
ExitNode(temp.id, new Action(
                                                      temp.id, W)));
                                    if (temp.down == -100)
                                          Exits_temp.add(new
ExitNode(temp.id, new Action(
                                                      temp.id, S)));
                                    if (temp.up == -100)
```

```
                                          Exits_temp.add(new
ExitNode(temp.id, new Action(
                                                temp.id, N)));
                                    if (temp.get == -100)
                                          Exits_temp.add(new
ExitNode(temp.id, new Action(
                                                temp.id, GET)));
                                    if (temp.put == -100)
                                          Exits_temp.add(new
ExitNode(temp.id, new Action(
                                                temp.id, PUT)));
                              }
                        }

                        Exits.add(e - 1, Exits_temp);
                        // System.out.println();
                        // System.out.println("Exits for e=1");
                        /*
                         * for (int i = 0; i < Exits.get(e-1).size(); i++) {
temp2 =
                         * Exits.get(e-1).get(i); System.out.println("state " +
temp2.s +
                         * " state " + temp2.a.s + " action " + temp2.a.a); }
                         */

                  } else {

                        ArrayList<ExitNode> Exits_temp = new
ArrayList<ExitNode>();// (s,a)
                        for (int i = 0; i < Graph.get(e - 1).size(); i++) {
                              temp = Graph.get(e - 1).get(i);
                              if (temp.id == -1) {
                                    for (int j = 0; j < Exits.get(e - 1 -
1).size(); j++) {// e3odoi

                                          // pr0igoumenou

                                          // level
                                    temp2 = Exits.get(e - 1 - 1).get(j);
                                    if (temp2.a.a == PUT) {//mono me put
prokaleitai e3odos sto level 2
                                          Exits_temp.add(new
ExitNode(temp.id, temp2.a));

                                    }
                              }
                        }

                        Exits.add(e - 1, Exits_temp);
                        // System.out.println();
                        // System.out.println("Exits for e=2");
                        /*
                         * for (int j = 0; j < Exits.get(e - 1).size(); j++) {
temp2 =
```

```
                    * Exits.get(e - 1).get(j); System.out.println("state "
+ temp2.s +
                    * " state " + temp2.a.s + " action " + temp2.a.a); }
                    */

              }

        }
/*evresi eisodwn tou epipedou e*/
        public static void findEntries(int e) {
              // To taxi exei opoiadipote tixaia arxiki 8esi sto state
space
              if (e == 1) {
                    Entries = X.get(e - 1);//(0..24)

              }

              else if (e == 2) {
                    int temp;
                    for (int i = 0; i < X.get(e - 1).size(); i++) {
                          temp = X.get(e - 1).get(i);
                          if (temp != -1)// den mporei na exi arxiki 8esi o
passenger sto
                                                // taxi(-1)
                                Entries.add(States.get(e - 1).get(i));// 0
4 20 23 (level 2)
                    }
              }
        }
/*epistrefei ton pinaka geitniasis tou level 1*/
        public static int[][] findAdjacentMatrix1(int sizex, int sizey,
                    ArrayList<Wall> walls) {
              boolean flag = false;
              int[][] table = new int[SA_Gridworld.size_x *
SA_Gridworld.size_y][SA_Gridworld.size_x
                          * SA_Gridworld.size_y];
              for (int i = 0; i < (SA_Gridworld.size_x *
SA_Gridworld.size_y); i++) {
                    for (int j = 0; j < (SA_Gridworld.size_x *
SA_Gridworld.size_y); j++) {
                          flag = false;
                          if (i == j) {
                                table[i][j] = 1;

                                continue;
                          }
                          for (int k = 0; k < SA_Gridworld.wall.size();
k++) {
                                Wall temp = SA_Gridworld.wall.get(k);
                                if (((temp.cell1 == i) && (temp.cell2 ==
j))
                                            || ((temp.cell1 == j) &&
(temp.cell2 == i))) {
                                      table[i][j] = 0;
                                      flag = true;
                                }
```

```
                        }
                        if ((!flag)
                                    && (((i + 1 == j) && (j %
SA_Gridworld.size_x != 0))
                                                || ((i - 1 == j) && (i %
SA_Gridworld.size_x != 0))
                                                || (i -
SA_Gridworld.size_x == j) || (i
                                                + SA_Gridworld.size_x ==
j))) {
                              table[i][j] = 1;

                        }

                  }
            }
            /*
             * for (int i=0;i<25;i++){ for (int j=0;j<25;j++){
             * System.out.print(table[i][j]); } System.out.println();}
             */

            System.out.println();
            return table;

      }
/*epistrefei to index tou komvou me vasi to id tou gia level e */
      public static int findIndex(int id, int e) {
            graph_node g;
            int t;
            for (t = 0; t < Graph.get(e - 1).size(); t++) {
                  g = Graph.get(e - 1).get(t);
                  if (g.id == id)
                        return g.index;
            }

            return -5;// se periptwsi pou einai e3odos (-100) na dinei
ton dikti -5
      }
      /*epistrefei ton pinaka geitniasis tou level 2*/
      public static int[][] findAdjacentMatrix2(int e) {
            int[][] table = new
int[SA_Gridworld.numOfStartPosition][SA_Gridworld.numOfStartPosition];
            graph_node temp;
            for (int i = 0; i < SA_Gridworld.numOfStartPosition; i++) {
                  for (int j = 0; j < SA_Gridworld.numOfStartPosition;
j++)
                        table[i][j] = 0;
            }

            for (int i = 0; i < SA_Gridworld.numOfStartPosition; i++) {

                  temp = Graph.get(e - 1).get(i);

                  if (!((temp.id == temp.right) || (temp.right == -100)
|| (temp.right == -1))) {
                        table[i][temp.right] = 1;
```

```
                } else if ((temp.right == -1))
                        table[i][0] = 1;

                else if (temp.right == temp.id)
                        table[i][i] = 1;

                if (!((temp.id == temp.left) || (temp.left == -100) ||
(temp.left == -1))) {
                        table[i][temp.left] = 1;
                } else if ((temp.left == -1))
                        table[i][0] = 1;

                else if (temp.left == temp.id)
                        table[i][i] = 1;

                if (!((temp.id == temp.up) || (temp.up == -100) ||
(temp.up == -1))) {
                        table[i][temp.up] = 1;
                } else if ((temp.up == -1))
                        table[i][0] = 1;

                else if (temp.up == temp.id)
                        table[i][i] = 1;

                if (!((temp.id == temp.down) || (temp.down == -100) ||
(temp.down == -1))) {
                        table[i][temp.down] = 1;
                } else if ((temp.down == -1))
                        table[i][0] = 1;

                else if (temp.down == temp.id)
                        table[i][i] = 1;

                if (!((temp.id == temp.get) || (temp.get == -100) ||
(temp.get == -1))) {
                        table[i][temp.get] = 1;
                } else if ((temp.get == -1))
                        table[i][0] = 1;

                else if (temp.get == temp.id)
                        table[i][i] = 1;

                if (!((temp.id == temp.put) || (temp.put == -100) ||
(temp.put == -1))) {
                        table[i][temp.put] = 1;
                } else if ((temp.put == -1))
                        table[i][0] = 1;

                else if (temp.put == temp.id)
                        table[i][i] = 1;

        }
        /*
         * for (int i=0;i<5;i++){ for (int j=0;j<5;j++)
         * System.out.print(table[i][j]);
         *
```

```
             * System.out.println();}
             */

            return table;
      }
      /*sinartsi Regions pou antistoixei sto 4evdokwdika tou
Hengst(sel.89), gia evresi twn MERs*/
      public static void Regions(int e) {

            ArrayList<Integer> Sccs = new ArrayList<Integer>(); // ta
diaforetika

                  // scc labels pou

                  // iparxoun
            int counter = 0, previous, in_r, in_l, in_u, in_d, in_g,
in_p;
            graph_node n;

            do {
                  if ((e == 1) && (counter == 0))// (counter==0) -> 1i
fora pou
                                                                  //
dimiourgeitai o pinakas
                                                                  //
geitniasis
                        adj = findAdjacentMatrix1(SA_Gridworld.size_x,
                              SA_Gridworld.size_y,
SA_Gridworld.wall);

                  else if ((e == 2) && (counter == 0))
                        adj = findAdjacentMatrix2(e);

                  else if (counter != 0) {

                        for (int i = 0; i < Graph.get(e - 1).size(); i++)
{
                              n = Graph.get(e - 1).get(i);
                              in_r = findIndex(n.right, e);
                              in_l = findIndex(n.left, e);
                              in_u = findIndex(n.up, e);
                              in_d = findIndex(n.down, e);
                              in_g = findIndex(n.get, e);
                              in_p = findIndex(n.put, e);

                              // epanaferei ton pinaka (tis allages pou
eginan stin grammi
                              // 7 tis regions)
                              if (n.right != -100)
                                    adj[n.index][in_r] = 1;

                              if (n.left != -100)
                                    adj[n.index][in_l] = 1;

                              if (n.up != -100)
                                    adj[n.index][in_u] = 1;
```

```
                        if (n.down != -100)
                              adj[n.index][in_d] = 1;

                        if (n.get != -100)
                              adj[n.index][in_g] = 1;

                        if (n.up != -100)
                              adj[n.index][in_u] = 1;

                  }

            }

            previous = Sccs.size();// 0 tin 1i fora
            if (e == 1)
                  SCC(SA_Gridworld.size_x * SA_Gridworld.size_y,
e);
            else
                  SCC(SA_Gridworld.numOfStartPosition, e);

            for (int i = 0; i < SCC.length; i++) {
                  if (Sccs.contains(SCC[i]))
                        continue;

                  else
                        Sccs.add(SCC[i]);

            }

            if ((Sccs.size() != 1) && (Sccs.size() - previous >
0))// 2nd part

                        // blocks

                        // duplicates

                        // entries

                        // in Exits

                        // & Entries
                  Connected_Sccs(e);

            counter++;

      } while (counter <= 1 || Sccs.size() - previous > 0);//
(telos repeat

                  // until 1-10)

      // Join SCCs into regions so that all region entry states can
reach all
      // exit states
      if (Sccs.size() != 1) {
            Join_Sccs(e);
```

```
                  // efoson i join_sccs enwse kapoia sccs prepei na
ipologsistei 3ana
                  // o arithmos twn diaforetikwn sccs
                  Sccs.clear();
                  for (int i = 0; i < SCC.length; i++) {
                        if (Sccs.contains(SCC[i]))
                              continue;

                        else
                              Sccs.add(SCC[i]);

                  }
            }
            create_Mer(e, Sccs.size());

      }
      /*dimiourgei ta MERs pou vrike*/
      public static void create_Mer(int e, int sccs_size) {

            ArrayList<MER> mer = new ArrayList<MER>();
            ArrayList<ExitNode> E = Exits.get(e - 1);
            mer.add(new MER(sccs_size, SCC, Entries, E));
            MERS.add(e - 1, mer);

      }
      /*antistoixei sti grammi 12 tis Regions tou Hengst (sel.89)*/
      public static boolean No_Other_Entry(scc_couples couple, int e) {
            // System.out.println("No other Entry");
            boolean flag1 = true, flag2 = true, flag3 = true, flag4 =
true, flag5 = true, flag6 = true;
            int num_of_scc = SCC[couple.node2];
            int right_index = -5, left_index = -5, up_index = -5,
down_index = -5, get_index = -5, put_index = -5;
            graph_node node, node2;

            for (int j = 0; j < Graph.get(e - 1).size(); j++) {
                  node = Graph.get(e - 1).get(j);
                  if (SCC[node.index] != num_of_scc) {
                        for (int k = 0; k < Graph.get(e - 1).size(); k++)
{
                              node2 = Graph.get(e - 1).get(k);

                              if (node.right == node2.id) {
                                    right_index = node2.index;
                              }
                              if (node.left == node2.id) {
                                    left_index = node2.index;
                              }
                              if (node.up == node2.id) {
                                    up_index = node2.index;
                              }
                              if (node.down == node2.id) {
                                    down_index = node2.index;
                              }
                              if (node.get == node2.id) {
                                    get_index = node2.index;
```

```
                                    }
                                    if (node.put == node2.id) {
                                          put_index = node2.index;
                                    }
                              }

                              // System.out.println("right_index
"+right_index+" left index "+left_index+" up_index "+up_index+" down
index "
                              // +
                              // +down_index+" get index "+get_index+"
put_index "+put_index);
                              // ean anikoun sto idio scc kai na min einai o
eautos tou
                              if ((right_index != -5) && (SCC[right_index] ==
num_of_scc)) {
                                    flag1 = false;
                                    // System.out.println(node.id+"me right
ENTRY");
                              }
                              if ((left_index != -5) && (SCC[left_index] ==
num_of_scc)) {
                                    flag2 = false;
                                    // System.out.println(node.id+"me left
ENTRY");
                              }
                              if ((up_index != -5) && (SCC[up_index] ==
num_of_scc)) {
                                    flag3 = false;
                                    // System.out.println(node.id+"me up
ENTRY");
                              }
                              if ((down_index != -5) && (SCC[down_index] ==
num_of_scc)) {
                                    flag4 = false;
                                    // System.out.println(node.id+"me down
ENTRY");
                              }
                              if ((get_index != -5) && (SCC[get_index] ==
num_of_scc)) {
                                    flag5 = false;
                                    // System.out.println(node.id+"me get
ENTRY");
                              }
                              if ((put_index != -5) && (SCC[put_index] ==
num_of_scc)) {
                                    flag6 = false;
                                    // System.out.println(node.id+"me put
ENTRY");
                              }

                        }
                  }
                  if ((flag1 == false) || (flag2 == false) || (flag3 == false)
                              || (flag4 == false) || (flag5 == false) || (flag6
== false)) {
```

```
                  // System.out.println("false");
                  return false;
            } else {
                  // System.out.println("true");
                  return true;
            }
            // checked
      }
      /*antistoixei sti grammi 12 tis Regions tou Hengst (sel.89)*/
      public static boolean No_Other_Exit(scc_couples couple, int e) {

            boolean flag1 = true, flag2 = true, flag3 = true, flag4 =
true, flag5 = true, flag6 = true;
            int num_of_scc = SCC[couple.node1];
            int right_index = -5, left_index = -5, up_index = -5,
down_index = -5, get_index = -5, put_index = -5;
            graph_node node, node2;

            for (int j = 0; j < Graph.get(e - 1).size(); j++) {
                  node = Graph.get(e - 1).get(j);
                  if (SCC[node.index] == num_of_scc) {// vriskw ton node1
kai ta nodes

      // pou metavainnei
                        for (int k = 0; k < Graph.get(e - 1).size(); k++)
{
                              node2 = Graph.get(e - 1).get(k);

                              if (node.right == node2.id) {
                                    right_index = node2.index;
                              }
                              if (node.left == node2.id) {
                                    left_index = node2.index;
                              }
                              if (node.up == node2.id) {
                                    up_index = node2.index;
                              }
                              if (node.down == node2.id) {
                                    down_index = node2.index;
                              }
                              if (node.get == node2.id) {
                                    get_index = node2.index;
                              }
                              if (node.put == node2.id) {
                                    put_index = node2.index;
                              }
                        }

                        // System.out.println("right_index
"+right_index+" left index "+left_index+" up_index "+up_index+" down
index "
                        // +
                        // +down_index+" get index "+get_index+"
put_index "+put_index);
                        // na min anikoun sto idio scc kai na min einai o
eautos tou
```

```
                        // tote iparxei perissoteres apo mia e3odoi
                        if ((right_index != -5) && (SCC[right_index] !=
num_of_scc)
                                    && (couple.action != E)) {
                              flag1 = false;

                        }
                        if ((left_index != -5) && (SCC[left_index] !=
num_of_scc)
                                    && (couple.action != W)) {
                              flag2 = false;

                        }
                        if ((up_index != -5) && (SCC[up_index] !=
num_of_scc)
                                    && (couple.action != N)) {
                              flag3 = false;

                        }
                        if ((down_index != -5) && (SCC[down_index] !=
num_of_scc)
                                    && (couple.action != S)) {
                              flag4 = false;

                        }
                        if ((get_index != -5) && (SCC[get_index] !=
num_of_scc)
                                    && (couple.action != GET)) {
                              flag5 = false;

                        }
                        if ((put_index != -5) && (SCC[put_index] !=
num_of_scc)
                                    && (couple.action != PUT)) {
                              flag6 = false;

                        }

                  }
            }
            // System.out.println("
"+(flag1||flag2||flag3||flag4||flag5||flag6));
            return (flag1 || flag2 || flag3 || flag4 || flag5 || flag6);
            // checked
      }
      /*antistoixei stis grammes 12-16 tis Regions tou Hengst (sel.89)*/
      public static void Join_Sccs(int e) {

            scc_couples couple;
            graph_node node;
            ExitNode exit = null;
            int num1 = 0, num2, index;
            boolean unique_exit, unique_entry;

            for (int i = 0; i < Connected_sccs.size(); i++) {// filagmena
ta scc
```

```
                    // couples me vasi

                    // indexes tous(oxi

                    // ta ids)
                    couple = Connected_sccs.get(i);

                    unique_exit = No_Other_Exit(couple, e);
                    unique_entry = No_Other_Entry(couple, e);
                    // System.out.println("couple "+couple.node1+"
"+couple.node2+" "+(unique_exit||unique_entry));

                    if (unique_exit || unique_entry) {
                          for (int k = 0; k < Graph.get(e - 1).size(); k++)
{
                                node = Graph.get(e - 1).get(k);
                                if (node.index == couple.node1)
                                      num1 = node.id;
                          }
                          for (int j = 0; j < Exits.get(e - 1).size(); j++)
{
                                exit = Exits.get(e - 1).get(j);

                                if ((exit.a.a == couple.action) && (exit.s
== num1))
                                      Exits.get(e - 1).remove(j);
                          }

                          for (int k = 0; k < Entries.size(); k++) {
                                int j = Entries.get(k);
                                index = findIndex(j, e);
                                if (index == couple.node2)
                                      Entries.remove(k);
                          }

                          num2 = SCC[couple.node2];
                          for (int w = 0; w < SCC.length; w++)
                                if (SCC[w] == num2)
                                      SCC[w] = SCC[couple.node1];

                          // System.out.println();

                    }

              }

              // System.out.println("SCCs");
              // for (int a = 0; a < SCC.length; a++)
              // System.out.print(" " + SCC[a] + " ");
              // System.out.println();
              /*
               * for (int j = 0; j < Exits.get(e - 1).size(); j++) {
ExitNode temp2 =
               * Exits.get(e - 1).get(j); System.out.println("state " +
temp2.s +
```

```
            * " state " + temp2.a.s + " action " + temp2.a.a); }
            *
            * for (int j = 0; j < Entries.size(); j++) { int i =
Entries.get(j);
            * System.out.println("entry " + i); }
            */

      }
      /*antistoixei stis grammes 6-9 tis Regions tou Hengst (sel.89)*/
      public static void Connected_Sccs(int e) {// code lines 6-9
            graph_node node1, node2;

            int index_left = -5, index_right = -5, index_up = -5,
index_down = -5, index_get = -5, index_put = -5;

            for (int i = 0; i < Graph.get(e - 1).size(); i++) {// -1 0 4
20 23
                  node1 = Graph.get(e - 1).get(i);
                  index_left = -5;
                  index_right = -5;
                  index_up = -5;
                  index_down = -5;
                  index_get = -5;
                  index_put = -5;

                  for (int j = 0; j < Graph.get(e - 1).size(); j++) {// -
1 0 4 20 23

                        node2 = Graph.get(e - 1).get(j);
                        // (2 different sccs ( && not exit implied ))
                        if ((node1.right == node2.id)) {
                              index_right = node2.index;

                              // System.out.println(" node "+node1.id+"
right="+node1.right+"    "+node1.index+"    "+index_right);
                        }
                        if ((node1.left == node2.id)) {
                              index_left = node2.index;

                              // System.out.println("node="+node1.id+"
left="+node1.left+"    "+node1.index+"    "+index_left);
                        }
                        if ((node1.up == node2.id)) {
                              index_up = node2.index;

                              // System.out.println("node="+node1.id+"
up="+node1.up+"    "+node1.index+"    "+index_up);
                        }
                        if ((node1.down == node2.id)) {
                              index_down = node2.index;

                              // System.out.println("node="+node1.id+"
down="+node1.down+"    "+node1.index+"    "+index_down);
                        }
                        if ((node1.get == node2.id)) {
                              index_get = node2.index;
```

```
                                // System.out.println("node="+node1.id+"
get="+node1.get+"    "+node1.index+"    "+index_get);
                        }
                        if ((node1.put == node2.id)) {
                                index_put = node2.index;

                                // System.out.println("node="+node1.id+"
put="+node1.put+"    "+node1.index+"    "+index_put);
                        }
                }

                if ((node1.right != -100) && (SCC[node1.index] !=
SCC[index_right])) { // oxi

                                                        // oi

                                                        // e3odoi
                        // System.out.println("different SCCs node1=" +
node1.id
                        // + " node2=" + node1.right + " me right");

                        adj[node1.index][index_right] = 0;
                        // System.out.print(" "+node1.index+"
"+index_right+" ="+adj[node1.index][index_right]+" ");

                        Exits.get(e - 1).add(
                                        new ExitNode(node1.id, new
Action(node1.id, E)));
                        Entries.add(node1.right);//
                        Connected_sccs
                                        .add(new scc_couples(node1.index, E,
index_right));// ta

                                                // indexes

                                                // tou

                                                // zevgous

                                                // h

                                                // Connected_sccs.add(new

                                                // scc_couples(node1.id,

                                                // E,node1.right));
                }
                if ((node1.left != -100) && (SCC[node1.index] !=
SCC[index_left])) { // oxi

                                                        // oi

                                                        // e3odoi
                        // System.out.println("different SCCs node1=" +
node1.id
```

```
                        // + " node2=" + node1.left + " me left");

                        adj[node1.index][index_left] = 0;

                        // System.out.print(" "+node1.index+"
"+index_left+" ="+adj[node1.index][index_left]+" ");
                        Exits.get(e - 1).add(
                                        new ExitNode(node1.id, new
Action(node1.id, W)));
                        Entries.add(node1.left);
                        Connected_sccs.add(new scc_couples(node1.index,
W, index_left));// ta

                                                        // indexes

                                                        // tou

                                                        // zevgous
                }
                if ((node1.up != -100) && (SCC[node1.index] !=
SCC[index_up])) { // oxi

                                                        // oi

                                                        // e3odoi
                        // System.out.println("different SCCs node1=" +
node1.id
                        // + " node2=" + node1.up + " me up");

                        adj[node1.index][index_up] = 0;

                        // System.out.print(" "+node1.index+"
"+index_up+" ="+adj[node1.index][index_up]+" ");
                        Exits.get(e - 1).add(
                                        new ExitNode(node1.id, new
Action(node1.id, N)));
                        Entries.add(node1.up);
                        Connected_sccs.add(new scc_couples(node1.index,
N, index_up));// ta

                                                        // indexes

                                                        // tou

                                                        // zevgous
                }
                if ((node1.down != -100) && (SCC[node1.index] !=
SCC[index_down])) { // oxi

                                                                // oi

                                                                // e3odoi
                        // System.out.println("different SCCs node1=" +
node1.id
                        // + " node2=" + node1.down + " me down");
```

```
                        adj[node1.index][index_down] = 0;

                        // System.out.print(" "+node1.index+"
"+index_down+" ="+adj[node1.index][index_down]+" ");
                        Exits.get(e - 1).add(
                                    new ExitNode(node1.id, new
Action(node1.id, S)));
                        Entries.add(node1.down);
                        Connected_sccs.add(new scc_couples(node1.index,
S, index_down));// ta

                                                // indexes

                                                // tou

                                                // zevgous
                    }

                    if ((node1.get != -100) && (SCC[index_get] !=
SCC[node1.index])) {// oxi

                                                // tis

                                                // e3odous
                        // System.out.println("different SCCs node1=" +
node1.id
                        // + " node2=" + node1.get + " me get");
                        adj[node1.index][index_get] = 0;
                        // System.out.print(" "+node1.index+"
"+index_get+" ="+adj[node1.index][index_get]+" ");
                        Exits.get(e - 1).add(
                                    new ExitNode(node1.id, new
Action(node1.id, GET)));
                        Entries.add(node1.get);
                        Connected_sccs
                                    .add(new scc_couples(node1.index,
GET, index_get));
                    }
                    if ((node1.put != -100) && (SCC[index_put] !=
SCC[node1.index])) {// oxi

                                                // tis

                                                // e3odous
                        // System.out.println("different SCCs node1=" +
node1.id
                        // + " node2=" + node1.put + " me put");
                        adj[node1.index][index_put] = 0;
                        // System.out.print(" "+node1.index+"
"+index_put+" ="+adj[node1.index][index_put]+" ");
                        Exits.get(e - 1).add(
                                    new ExitNode(node1.id, new
Action(node1.id, PUT)));
                        Entries.add(node1.put);
                        Connected_sccs
```

```
                                    .add(new scc_couples(node1.index,
PUT, index_put));
                  }

            }

            // checked
      }
      /*antistoixei stis grammes 13-19 tis SCC tou Hengst (sel.213)*/
      public static void DFS1(int s, int states) {

            col[s] = 'G';
            finTime++;
            for (int j = 0; j < states; j++) {

                  if ((adj[s][j] == 1) && (col[j] == 'W')) {
                        DFS1(j, states);
                  }
            }
            col[s] = 'B';
            f[s] = finTime;
            finTime++;

      }
      /*antistoixei stis grammes 20-24 tis SCC tou Hengst (sel.213)*/
      public static void DFS2(int s, int states) {
            col[s] = 'G';
            for (int j = 0; j < states; j++) {

                  if ((adj[j][s] == 1) && (col[j] == 'W')) {
                        DFS2(j, states);
                  }
            }
            col[s] = 'B';
            SCC[s] = SCClabel;

      }
      /*evresi twn abstract actions tou epomenou epipedou me vasi tis
e3odous tou trexontos*/
      public static void nextLevelAbstractActions(int e) {// abstract
actions are

            // encoded in the same

            // way as exits.
            ExitNode exit;
            ArrayList<ExitNode> Actions_temp = new ArrayList<ExitNode>();
            for (int i = 0; i < Exits.get(e - 1).size(); i++) {
                  exit = Exits.get(e - 1).get(i);
                  Actions_temp.add(new ExitNode(exit.s, exit.a));

            }
            Actions.add(Actions_temp.size());
            Abstract_Actions.add(e - 1, Actions_temp);// abstract actions
tou level
```

```
	// 2 kai 3 einai oi e3odoi

	// tou epipedou 1 k 2
		// System.out.println("s="+Abstract_Actions.get(0).get(0).s+"
s="+Abstract_Actions.get(0).get(0).a.s+"
a="+Abstract_Actions.get(0).get(0).a.a);

	}
	/*grammi 3 tou HexQ sto Hengst (2002)*/
	public static void initializeA() {

		for (int i = 0; i < SA_Gridworld.actions; i++)
			Primitive_Actions.add(i);
		Actions.add(Primitive_Actions.size());

	}
	/*grammi 1 tou HexQ sto Hengst (2002)*/
	public static void initializeX() {
		ArrayList<Integer> X1 = new ArrayList<Integer>();
		ArrayList<Integer> X2 = new ArrayList<Integer>();
		ArrayList<Integer> X3 = new ArrayList<Integer>();
		int size;
		// initialize state variable taxi location
		for (int i = 0; i < SA_Gridworld.size_x *
SA_Gridworld.size_y; i++)
			// 0-25
			X1.add(i);

		size = SA_Gridworld.getNumOfStartPosition();
		// initialize state variable passenger location
		for (int i = 0; i < size; i++) {
			X2.add(SA_Gridworld.getPasLoc(i));// -1 0 4 20 23
			// System.out.print(" "+SA_Gridworld.getPasLoc(i)+" ");
		}
		// initialize state variable destination
		size = SA_Gridworld.getNumOfGoals();
		for (int i = 0; i < size; i++)
			X3.add(SA_Gridworld.getDestLoc(i));// 0 4 20 23

		X.add(X1);
		X.add(X2);
		X.add(X3);
	}
	/*grammi 2 tou HexQ sto Hengst (2002)*/
	public static void initializeS(int e) {
		ArrayList<Integer> temp = new ArrayList<Integer>();
		for (int i = 0; i < X.get(e - 1).size(); i++) {
			temp.add(X.get(e - 1).get(i));
		}
		States.add(e - 1, temp);
		// System.out.println("States for e="+e);

		// for(int i=0;i<States.get(e-1).size();i++){ int
		// j=States.get(e-1).get(i); System.out.print(" "+j+" "); }
```

```
}
/*i sinartisi SCC antistoixei se auti tou Hengst (2002) sel.213*/
public static void SCC(int states, int e) {
	finTime = 0;
	SCClabel = 0;
	int pos = -5, count = 0;
	int tab[] = new int[states];
	col = new char[states];
	f = new int[states];
	SCC = new int[states];

	for (int i = 0; i < states; i++) {
		col[i] = 'W';
		f[i] = Integer.MAX_VALUE;
		SCC[i] = Integer.MAX_VALUE;

	}

	for (int i = 0; i < states; i++) {
		if (col[i] == 'W') {
			DFS1(i, states);
		}

	}
	for (int j = 0; j < states; j++) {
		col[j] = 'W';
	}
	for (int i = 0; i < states; i++) {
		tab[i] = 0;
	}
	int max;
	for (int k = 0; k < states; k++) {
		max = Integer.MIN_VALUE;

		for (int l = 0; l < states; l++) {

			if (tab[l] == 1)
				continue;

			if ((f[l] > max) || (count == states - 1)) {
				max = f[l];
				pos = l;
			}
		}
		tab[pos] = 1;
		count++;
		// System.out.print(" "+f[pos]+" ");
		if (col[pos] == 'W') {
			DFS2(pos, states);
		}

		SCClabel++;// number of sccs
	}
	// System.out.println();
```

```
            // System.out.println("Scc label for e="+e+" ----->"+
SCClabel);
            // for(int j=0;j<SCC.length;j++)
            // System.out.print(" "+SCC[j]+" ");

      }

      /*sinartisi i opoia vriskei ta sub-MDPs olwn twn epipedwn, grammi 7
tou HexQ ston Hengst (2002)*/
      public static void constructSubMdps(int e) {

            sub_mdp m;
            boolean exist = false;
            ExitNode exit;
            graph_node x;
            int dif_states = 0, states = getStates(e), actions;
            if (e == 1) {

                  actions = SA_Gridworld.actions;
                  // System.out.println("Ta  actions tou level
1="+actions+"kai ta states="+states);
            } else {

                  actions = Exits.get(e - 2).size();// ta exits tou
proigoumenou

      // epipedou (1)
                  // System.out.println("Ta abstract actions tou level
2="+actions+"kai ta states="+states);
            }

            double[][] q = initializeQ(states, actions);
            // System.out.println(states+" "+actions+" "+e);
            /*
             * for (int i=0;i<states;i++){ for(int j=0;j<actions;j++)
             * System.out.print(" "+q[i][j]+" "); System.out.println();}
             */

            int[] visited = new int[Exits.get(e - 1).size()];// tosa sub-
mdps osa

                  // kai ta

                  // diaforetika exit

                  // states
            ArrayList<graph_node> trans = new ArrayList<graph_node>();//
transition

                              // model tou

                              // ka8e

                              // sub-mdp
            ArrayList<ExitNode> ex = new ArrayList<ExitNode>();// exits
tou ka8e
```

```
                  // sub-mdp

            ArrayList<sub_mdp> subs = new ArrayList<sub_mdp>();// ola ta
sub-mdp tou

                  // epipedou e

            for (int j = 0; j < visited.length; j++)
                  visited[j] = -5;
            // oi epomenes 12 grammes kwdika gia na vrw ton ari8mo twn
diaforetikwn
            // exit states tou transition model
            for (int i = 0; i < Exits.get(e - 1).size(); i++) {
                  exit = Exits.get(e - 1).get(i);
                  exist = false;
                  for (int j = 0; j < visited.length; j++)
                        if (visited[j] == exit.a.s)// an exit tou sub-mdp
exei idi
                                                                        //
kataxwri8ei
                              exist = true;
                  if (!exist) {
                        dif_states++;// ari8mos diaforetikwn exits states
                        for (int j = 0; j < visited.length; j++)
                              if (visited[j] == -5) {// kataxwrw to exit
tou sub-mdp stin
                                                                        // 1i
adeia 8esi tou pinaka
                                    visited[j] = exit.a.s;
                                    break;
                              }
                  }

            }// checked (0 4 20 23)

            for (int i = 0; i < dif_states; i++) {// gia ka8e exit
diaforetiko exit

      // state dimiourgw sub-mdp

                  trans = new ArrayList<graph_node>();
                  ex = new ArrayList<ExitNode>();

                  for (int j = 0; j < Graph.get(e - 1).size(); j++) {//
copy to

                        // transition

                        // model tou

                        // epipedou e
                        trans.add(new graph_node(Graph.get(e -
1).get(j).id, Graph.get(
                                    e - 1).get(j).up, Graph.get(e -
1).get(j).down, Graph
```

```
                                    .get(e - 1).get(j).right, Graph.get(e
- 1).get(j).left,
                                    Graph.get(e - 1).get(j).get,
                                    Graph.get(e - 1).get(j).put,
                                    Graph.get(e - 1).get(j).index));

                  }

                  for (int j = 0; j < Exits.get(e - 1).size(); j++) {
                        exit = Exits.get(e - 1).get(j);
                        if (visited[i] == exit.a.s)
                              ex.add(exit);// i monadiki e3odos tou ka8e
sub-mdp
                  }

                  exit = ex.get(0);// ena exit state (or mutiples if
multiple actions
                                                // exists)
                  // afairw apo to transition model tou sub-mdp tis
ipoloipes
                  // e3odous,afinnw mono tin diki tou
                  for (int j = 0; j < trans.size(); j++) {

                        x = trans.get(j);

                        if (x.id != exit.s) {
                              if (x.up == -100)
                                    x.up = x.id;
                              if (x.down == -100)
                                    x.down = x.id;
                              if (x.right == -100)
                                    x.right = x.id;
                              if (x.left == -100)
                                    x.left = x.id;
                              if (x.get == -100)
                                    x.get = x.id;

                              if (x.put == -100)
                                    x.put = x.id;

                        }
                  }

                  m = new sub_mdp(i, e, trans, ex, q);

                  subs.add(m);

            }

            Sub_Mdps.add(e - 1, subs);
            for (int i = 0; i < Sub_Mdps.get(e - 1).size(); i++) {
                  m = Sub_Mdps.get(e - 1).get(i);
                  // System.out.println("e="+e+" ");
                  /*
                   * for (int j =0;j<m.exits.size();j++){
exit=m.exits.get(j);
```

```
                     * System.out.println(exit.s+" "+exit.a.s+"
"+exit.a.a);
                     *
                     * }
                     */
                    /*
                     * for (int j =0;j<m.graph.size();j++){
x=m.graph.get(j);
                     * System.out.println(" node= " + x.id + " up= " + x.up
+ " down= "
                     * + x.down + " right= " + x.right + " left= " + x.left
+ " get=" +
                     * x.get + " put=" + x.put + " index=" + x.index);
                     *
                     * }
                     */

              }

       }
       /*dimiourgia tou top-level sub-mdp (den iparxoun e3odoi se auto
alla oute kai transition model)*/
       public static void constructTopLevelSub(int e) {
              ArrayList<sub_mdp> S = new ArrayList<sub_mdp>();

              sub_mdp sub = null;
              int actions = getActions(e);

              double E[][] = new double[SA_Gridworld.numOfGoals][actions];
              E = initializeQ(SA_Gridworld.numOfGoals, actions);
              sub = new sub_mdp(0, e, null, null, E);
              S.add(sub);
              Sub_Mdps.add(S);

       }
       /*reward function gia sinartisi Execute, grammi 12 tis Execute
selida 99 tou Hengst(2002)*/
       public static re_state takeAction(int action, State_Variables s) {
              re_state next=null;
              int reward = 0;
              int taxi = s.taxi;
              int passenger = s.passenger;
              int destination = s.destination;
              // int reward = -1;//for each action
              switch (action) {
              // north (move up)
              case 0: // ektos orion tou grid
                     if ((!((s.taxi - SA_Gridworld.size_x) < 0))
                                   && (!(SA_Gridworld.checkForWall(s.taxi,
                                                 (s.taxi -
SA_Gridworld.size_x))))) {
                            taxi = s.taxi - SA_Gridworld.size_x;
                            reward = SA_Gridworld.simple_move;
                     } else {
                            reward = SA_Gridworld.wall_hit;
                     }
```

```
                  break;
            // south (move down)
            case 1: // ektos orion tou grid
                  if ((!((s.taxi + SA_Gridworld.size_y) >=
(SA_Gridworld.size_x * SA_Gridworld.size_y)))
                              && (!(SA_Gridworld.checkForWall(s.taxi,
                                          (s.taxi +
SA_Gridworld.size_y))))) {// na min

                        // iparxei wall
                        taxi = s.taxi + SA_Gridworld.size_y;
                        reward = SA_Gridworld.simple_move;
                  } else {
                        reward = SA_Gridworld.wall_hit;
                  }
                  break;
            // east (move right)
            case 2: // hit the wall or ektos orion tou grid !!!!!
                  if ((!((s.taxi % SA_Gridworld.size_y) ==
(SA_Gridworld.size_y - 1)))
                              && (!(SA_Gridworld.checkForWall(s.taxi,
(s.taxi + 1))))) {
                        taxi = s.taxi + 1;
                        reward = SA_Gridworld.simple_move;
                  } else {
                        reward = SA_Gridworld.wall_hit;
                  }
                  break;
            // west (move left)
            case 3: // hit the wall or ektos orion tou grid
                  if ((!((s.taxi % SA_Gridworld.size_y) == 0))
                              && (!(SA_Gridworld.checkForWall(s.taxi,
(s.taxi - 1))))) {// !!!!!!
                        taxi = s.taxi - 1;
                        reward = SA_Gridworld.simple_move;
                  } else {
                        reward = SA_Gridworld.wall_hit;
                  }
                  break;
            // pickup the passenger
            case 4: // location without a passenger or passenger already
in the taxi
                  if ((s.taxi != SA_Gridworld.getPasLoc(s.passenger))
                              || (SA_Gridworld.getPasLoc(s.passenger) ==
-1)) {
                        reward = SA_Gridworld.wrong_get;
                  } else {
                        passenger = 0;
                        reward = SA_Gridworld.simple_move;
                  }
                  break;
            // put down the passenger
            case 5: // wrong destination or passenger not in the taxi
                  if ((s.taxi != SA_Gridworld.getDestLoc(s.destination))
```

```
                              || (SA_Gridworld.getPasLoc(s.passenger) !=
-1)) {
                        reward = SA_Gridworld.wrong_put;
                  } else { // Successful passenger delivery
                        passenger = s.destination + 1;
                        reward = SA_Gridworld.success;
                  }
                  break;
            default:
                  System.err.println("Invalid number for
action..Exiting..");
                  System.exit(-1);
            }
            next=new re_state(reward,new
State_Variables(taxi,passenger,destination));

            return next;
      }


      /*pseudoreward function grammi 7 sel.18 tou Hengst(2002), gia xrisi
sti sinartisi ValueIteration */
      public static next_state simple_TakeAction(int action, int s) {
      int new_s=s;
      int reward = -5;
      next_state next=null;
            switch (action) {
            // north (move up)
            case 0: // ektos orion tou grid
                  if ((!((s - SA_Gridworld.size_x) < 0))
                              && (!(SA_Gridworld.checkForWall(s,
                                          (s - SA_Gridworld.size_x))))) {
                        new_s= s - SA_Gridworld.size_x;
                        reward = SA_Gridworld.simple_move;
                  } else {
                        reward = SA_Gridworld.wall_hit;

                  }

                  break;
            // south (move down)
            case 1: // ektos orion tou grid
                  if ((!((s + SA_Gridworld.size_y) >=
(SA_Gridworld.size_x * SA_Gridworld.size_y)))
                              && (!(SA_Gridworld.checkForWall(s,
                                          (s + SA_Gridworld.size_y)))))
{// na min

                        // iparxei wall
                        new_s = s + SA_Gridworld.size_y;
                        reward = SA_Gridworld.simple_move;
                  } else {
                        reward = SA_Gridworld.wall_hit;
                  }
                  break;
            // east (move right)
```

```
            case 2: // hit the wall or ektos orion tou grid !!!!!
                  if ((!((s % SA_Gridworld.size_y) ==
(SA_Gridworld.size_y - 1)))
                              && (!(SA_Gridworld.checkForWall(s, (s +
1))))) {
                        new_s = s + 1;
                        reward = SA_Gridworld.simple_move;
                  } else {
                        reward = SA_Gridworld.wall_hit;
                  }
                  break;
            // west (move left)
            case 3: // hit the wall or ektos orion tou grid
                  if ((!((s % SA_Gridworld.size_y) == 0))
                              && (!(SA_Gridworld.checkForWall(s, (s -
1))))) {// !!!!!!
                        new_s = s - 1;
                        reward = SA_Gridworld.simple_move;
                  } else {
                        reward = SA_Gridworld.wall_hit;
                  }
                  break;
            // pickup the passenger
            case 4: // location without a passenger or passenger already
in the taxi

                        reward = SA_Gridworld.wrong_get;
                  break;
            // put down the passenger
            case 5: // wrong destination or passenger not in the taxi

                        reward = SA_Gridworld.wrong_put;

                  break;
            default:
                  System.err.println("Invalid number for
action..Exiting..");
                  System.exit(-1);
            }
            next=new next_state(new_s,reward);

            return next;
      }
      /*antistoixei sti sinartisi ValueIteration sto Hengst (2002),
selida 18*/
      public static void valueIteration(sub_mdp m, State_Variables state,
int e) {

            int reward = -1, states, actions, i = 0, ii = 0, next_st = 0;
            double q, V, D;
            next_state next;
            ExitNode ex, abs, m_ex;
            State_Variables temp = new State_Variables(0, 0, 0);

            states = getStates(e);
            actions = Actions.get(e - 1);
```

```
            while (true) {
                  ii++;
                  D = 0;
                   if (e==2)
                  System.out.println("trial"+ii);
                  for (int s = 0; s < states; s++) {// tou sub_mdp
                        for (int a = 0; a < actions; a++) {
                              q = m.E[s][a];

                              if (e == 1) {// oloswsto
                                    next = simple_TakeAction(a, s);// ok
                                    temp.taxi = next.next_state;// ok
                                    reward = next.reward;// ok
                                    for (i = 0; i < m.exits.size(); i++)
{
                                          ex = m.exits.get(i);
                                          if ((ex.s == s) && (ex.a.a ==
a)) {
                                                reward = 0;
                                                break;
                                          }
                                    }
                              } else {
                                    abs = Abstract_Actions.get(e -
2).get(a);
                                    m_ex = m.exits.get(0);// exoun mono
mia e3odo ta

      // sub-mdps tou 2ou epipedou
                                    next_st =
SA_Gridworld.getPasLoc(s);// ok
                                    if (m_ex.s ==
SA_Gridworld.getPasLoc(s)
                                                && m_ex.a.s == abs.s// ok
to if {}
                                                && m_ex.a.a == abs.a.a) {
                                          reward = 0;
                                          next_st = abs.s;
                                    } else {
                                          if ((s == 0 && abs.a.a ==
GET)||(s!=0&&abs.a.a==GET&&abs.s!=SA_Gridworld.getPasLoc(s)))//2nd part
kanei get allou
                                                reward =
SA_Gridworld.wrong_get;
                                          else if (s != 0 && abs.a.a ==
PUT)
                                                reward =
SA_Gridworld.wrong_put;
                                          else if (s == 0 && abs.a.a ==
PUT) {//afinei ton epivati ekei pou prepei
                                                reward = -1;
                                                next_st = abs.s;
                                          }
                                          else if
(s!=0&&abs.a.a==GET&&abs.s==SA_Gridworld.getPasLoc(s)){//paralamvanei ton
epivati ap ekei pou vrisketai
```

```
                                                reward=-1;
                                                next_st=-1;
                                          }
                                    }


                                    for (int w=0;w<States.get(e-
1).size();w++){
                                          int st=States.get(e-1).get(w);
                                          if (next_st==st){//-1 0 4 20 23
                                                temp.passenger=w;
                                                break;}
                                          }

                                          temp.taxi=abs.s;//ok

                                    }

                                    V = Value(e, m, temp).v;// ok

                                    m.E[s][a] = (reward + (discount_factor *
V));// ok
                                    D = Math.max(D, Math.abs(q - m.E[s][a]));//
ok

                              }
                        }

                        if (D < 0.0001)
                              break;
                  }

            }
            /*antistoixei sti sinartisi Value sto Hengst (2002), selida 96*/
            public static val Value(int e, sub_mdp m, State_Variables s) {
                  val node = null;
                  if (e == 0) {
                        node = new val(0, 0);
                        return node;
                  }

                  double v = Double.NEGATIVE_INFINITY;//
                  double new_v = 0, max = 0;
                  int best_act = -5,best_act_1=-5,best_act_2=-5,best_act_3=-5;
                  sub_mdp sub;

                  if (e == 1) {
                        max = m.E[s.taxi][0];
                        best_act_1 = 0;
                        for (int ii = 1; ii < Primitive_Actions.size(); ii++) {
                              if (m.E[s.taxi][ii] > max) {
                                    max = m.E[s.taxi][ii];
                                    best_act_1 = ii;
                              }// to idio ka8e fora
                        }
```

```
            }

            else if (e == 2) {
                  max = m.E[s.passenger][0];
                  best_act_2 = 0;
                  for (int ii = 1; ii < Abstract_Actions.get(m.level -
2).size(); ii++) {
                        if (m.E[s.passenger][ii] > max){
                              max = m.E[s.passenger][ii];
                        best_act_2=ii; }
                  }
            }

            else if (e == 3) {
                  max = m.E[s.destination][0];
                  best_act_3 = 0;
                  for (int ii = 1; ii < Abstract_Actions.get(m.level -
2).size(); ii++) {
                        if (m.E[s.destination][ii] > max){
                              max = m.E[s.destination][ii];
                              best_act_3=ii;}
                  }
            }
            for (int i = 0; i < Actions.get(m.level - 1); i++) {

                  sub = findLowestSubMdp(m.level, i);

                  new_v = Value(e - 1, sub, s).v + max;
                  if (new_v > v) {
                        v = new_v;
                        if (e==1)
                        best_act=best_act_1;
                        else if(e==2)
                              best_act=best_act_2;
                        else
                              best_act=best_act_3;
                  }
                  //
            }


            node = new val(v, best_act);

            return node;

      }
      /*antistoixei sti sinartisi Execute sto Hengst (2002), selida 99*/
      public static void execute(int e, State_Variables s, int action) {

            boolean flag=false;
            int lse, a = 0;
            val pair = null, pair2 = null;
            sub_mdp m = null;
            ExitNode abs_act = null,exit=null;
            state=new State_Variables(s.taxi,s.passenger,s.destination);
            re_state next=null;
```

```
            if (e == 0) {
wr1.println("t="+state.taxi+" p="+state.passenger+"
d="+state.destination);
                  //epsilon = epsilon * decay;

                  next = takeAction(action, s);
                  reward=next.reward;
                  state=next.state;
                  wr1.println("action="+action+" reward="+reward+"
"+"t="+state.taxi+" p="+state.passenger+" d="+state.destination);
                  steps++;
                  sum_rewards+=reward;

                  return;
            }
            //-------------------------------------------------------
--------
            m = get_Sub_mdp(action, e);
            wr1.print("  sub="+m.id+" at e="+m.level);


            while (true) {

                  pair = Value(e, m, state);
                  if (r.nextDouble() < epsilon) {
                        //System.out.println("greedy action");
                        if (e == 1)
                              a = r.nextInt(Primitive_Actions.size());

                        else if (e == 2 || e == 3)
                              a = r.nextInt(Abstract_Actions.get(e -
2).size());
                  } else
                        a = pair.a;
                  //-----------------------------------------------------
--------------

                  if (e == 1)
                        lse = state.taxi;
                  else if (e == 2)
                        lse = state.passenger;
                  else
                        lse = state.destination;

                  execute(e - 1, state, a);
                  //-----------------------------------------------------
--------------

                  if (e==1){

                  for (int i=0;i<m.exits.size();i++){
                        exit=m.exits.get(i);
                        if
((exit.s==state.taxi&&exit.s==SA_Gridworld.getPasLoc(state.passenger)&&ex
it.a.a==a)
```

```
      ||(exit.s==state.taxi&&state.passenger==0&&exit.a.a==a)){
                              flag=true;
                              //System.out.println("E3odos 1");
                              wr1.println("E3ODOS 1");
                              break;
                        }

                  }
                  if (flag==true){
                        flag=false;
                  return;}
                  else{
                        pair2=Value(e,m,s);
                        m.E[lse][a]=(1-
learning_rate)*m.E[lse][a]+learning_rate*(reward+pair2.v);
                  }
                  }
                  else if (e==2){
                        for (int i=0;i<m.exits.size();i++){
                              exit=m.exits.get(i);
                              if
(exit.s==SA_Gridworld.getPasLoc(state.passenger)&&state.taxi==exit.a.s&&e
xit.a.a==a){
                                    flag=true;
                                    break;
                              }


                        }
                        if (flag==true){
                              flag=false;
                              //System.out.println("E3odos 2");
                              wr1.println("E3ODOS 2");
                              return;
                        }
                        else{
                              pair2=Value(e,m,state);
                              m.E[lse][a]=(1-
learning_rate)*m.E[lse][a]+learning_rate*(reward+pair2.v);
                        }
                  }
                  else{
                        if
(SA_Gridworld.getPasLoc(state.passenger)==SA_Gridworld.getDestLoc(state.d
estination)

      &&state.taxi==SA_Gridworld.getDestLoc(state.destination)){
                              //System.out.println("E3odos 3");
                              wr1.println("E3ODOS 3");
                              return;}
                        else{
                              pair2=Value(e,m,state);
                              m.E[lse][a]=(1-
learning_rate)*m.E[lse][a]+learning_rate*(reward+pair2.v);
                        }
```

```
                }


            }// while(true)

      }
      /*antistoixei sti grammi 4 tis sinartisis Execute sto Hengst
(2002), selida 99*/
      public static sub_mdp get_Sub_mdp(int action, int e) {
            if (e == 1) {
                  if (action == 0 || action == 1)
                        return Sub_Mdps.get(e - 1).get(0);
                  else if (action == 2 || action == 3)
                        return Sub_Mdps.get(e - 1).get(1);
                  else if (action == 4 || action == 5)
                        return Sub_Mdps.get(e - 1).get(2);
                  else if (action == 6 || action == 7)
                        return Sub_Mdps.get(e - 1).get(3);

            }
            //System.out.println("level="+e+" action="+action);
            return Sub_Mdps.get(e - 1).get(action);

      }//checked
      /*antistoixei sti grammi 6 tis sinartisis value sto Hengst (2002),
selida 96*/
      public static sub_mdp findLowestSubMdp(int e, int action) {

            if (e == 1)
                  return null;
            else if (e == 2) {
                  if (action == 0 || action == 1)
                        return Sub_Mdps.get(e - 2).get(0);
                  else if (action == 2 || action == 3)
                        return Sub_Mdps.get(e - 2).get(1);
                  else if (action == 4 || action == 5)
                        return Sub_Mdps.get(e - 2).get(2);
                  else if (action == 6 || action == 7)
                        return Sub_Mdps.get(e - 2).get(3);

            } else
                  return Sub_Mdps.get(e - 2).get(action);
            return null;

      }//checked

      public static void main(String[] args) {
            // TODO Auto-generated method stub
            try {
                  wr1=new PrintWriter("test.txt");
            } catch (FileNotFoundException e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();
            }
            int d, e;
```

```
		double timer;
		State_Variables s;
		sub_mdp m;

		set_Episode_Steps();
		set_Avg_Steps_Matrix();
		for (int i = 0; i < episodes; i++) {
			timer = (double) System.currentTimeMillis() / 1000;
			e = 1;
			do {
				s = new State_Variables(SA_Gridworld.size_x,
						SA_Gridworld.size_y,
SA_Gridworld.numOfStartPosition,
						SA_Gridworld.numOfGoals);
			} while (s.passenger==0);

			initializeX();// ok
			setDepth();// katw apo initializeX
			d = getDepth();
			initializeS(e); // ok
			initializeA();// ok

			// loop
			for (e = 1; e < d; e++) {
				initializeS(e + 1);// ok
				findEntries(e);// ok
				findTransitionModel(SA_Gridworld.size_x,
SA_Gridworld.size_y,
						SA_Gridworld.wall, e);// ok
				findExits(e);// ok
				Regions(e);// ok
				constructSubMdps(e);
				for (int j=0;j<Sub_Mdps.get(e-1).size();j++){
					m=Sub_Mdps.get(e-1).get(j);

					//valueIteration(m,s,e); infinite loop


				}
				nextLevelAbstractActions(e);// ok
				Entries.clear();
			}	// end of loop

			constructTopLevelSub(d);//sto anwtato sub-mdps den
iparxei e3odos alla termatiki sin8iki
			System.out.println("taxi="+s.taxi+"
pass="+s.passenger+" dest="+s.destination);
			execute(d, s, 0);

			Trials.addTimerPerEpisode(i,((double)
System.currentTimeMillis() / 1000 - timer));
			episodeSteps[i] = steps;
			episodeRewards[i]=sum_rewards;
			// System.out.println(episodeSteps[i]);
			steps = 0;
			sum_rewards=0;
```

```
                        X.clear();
                        States.clear();
                        Entries.clear();
                        Exits.clear();
                        MERS.clear();
                        Abstract_Actions.clear();
                        Primitive_Actions.clear();
                        Graph.clear();
                        Connected_sccs.clear();
                        Sub_Mdps.clear();
                        Actions.clear();

                  }
                  Trials.addTrialSteps(episodeSteps);
                  Trials.addTrialTime();
                  Trials.addRewards(episodeRewards);

            }

}

public class Action {
      int s;
      int a;
      Action(int s,int a) {
            this.s = s;
            this.a=a;
      }
}



public class ExitNode {
      int s;
      Action a;
      ExitNode(int s,Action a) {
            this.s = s;
            this.a=a;
      }
}



public class graph_node {

      int id;
      int up;
      int down;
      int right;
      int left;
      int get;
      int put;
      int index;
```

```
      graph_node(int id, int up, int down, int right, int left,int
get,int put,int index) {
            this.id = id;
            this.up = up;
            this.down = down;
            this.right=right;
            this.left=left;
            this.get=get;
            this.put=put;
            this.index=index;
      }


}


import java.util.ArrayList;


public class MER {
int num_of_sccs;
int [] SCC;
ArrayList<Integer> Entries = new ArrayList<Integer>();
 ArrayList<ExitNode> Exits = new ArrayList<ExitNode>();
 MER(int num_of_sccs,int [] SCC,ArrayList<Integer>
Entries,ArrayList<ExitNode> Exits){
       this.num_of_sccs=num_of_sccs;
       this.SCC=SCC;
       this.Entries=Entries;
       this.Exits=Exits;
 }
}



public class next_state {
int next_state;
int reward;
next_state(int next_state,int reward ){
      this.next_state=next_state;
      this.reward=reward;
}
}



public class re_state {
int reward;
State_Variables state;
re_state(int reward, State_Variables state){
      this.reward=reward;
      this.state=state;
}}


public class scc_couples {
```

```
int node1;
int action;
int node2;
scc_couples(int node1,int action,int node2 ){
      this.node1=node1;
      this.action=action;
      this.node2=node2;
}
}


import java.util.ArrayList;


public class sub_mdp {
      int id;
      int level;
      ArrayList<graph_node> graph=new ArrayList<graph_node>();
      ArrayList<ExitNode> exits=new ArrayList<ExitNode>();
      double E[][];
       sub_mdp(int id,int level,ArrayList<graph_node>
graph,ArrayList<ExitNode> exits,double E[][]){
            this.id=id;
            this.level=level;
            this.graph=graph;
            this.exits=exits;
            this.E=E;

       }



}

public class val {
double v;
int a;
val(double v, int a){
      this.v=v;
      this.a=a;
}
}
```

```java
/**
 * Voi8itiki klasi i opoia elegxei ti roi tou programmatos, kalei ton
HexQ gia to Taxi Problem
 *
 * @author Maria Kalli
 *
 */
import java.io.FileNotFoundException;
import java.io.PrintWriter;


public class Trials {
	public static int[][]matrix;
	public static double[][]matrix2;
	public static int[][]matrix3;
	public static double [] AverageStepsPerEpisode;
	public static double [] AverageTimePerEpisode;
	public static double [] timerPerEpisode;
	public static double [] AverageRewardPerEpisode;
	static int i;

	 static PrintWriter wr2;
	 static PrintWriter wr3;
	 static PrintWriter wr4;
	/**
	 * @param args
	 */
	 /*sinartisi pou orizei ta steps tou trial ka8e epeisodiou*/
	public static void addTrialSteps(int [] episodeSteps){
		for (int j=0;j<HexQ.episodeSteps.length;j++){
		matrix[j][i]=HexQ.episodeSteps[j];
		}
	}
	/*sinartisi pou orizei to xrono tou trial */
	public static void addTrialTime(){
		for (int j=0;j<HexQ.episodeSteps.length;j++){
			matrix2[j][i]=timerPerEpisode[j];
			}
	}
	/*sinartisi pou orizei ta rewards ka8e epeisodiou*/
	public static void addRewards(int [] episodeRewards){
		for (int j=0;j<HexQ.episodeRewards.length;j++){
			matrix3[j][i]=HexQ.episodeRewards[j];
			}
	}
	/*sinartisi pou dimiourgei tous v0i8itikous pinakes */
	public static void createMatrixes(){
		matrix=new int[HexQ.episodes][HexQ.trials];
		matrix2=new double[HexQ.episodes][HexQ.trials];
		matrix3=new int[HexQ.episodes][HexQ.trials];
		timerPerEpisode=new double [HexQ.episodes];
		AverageTimePerEpisode=new double[HexQ.episodes];
		AverageRewardPerEpisode=new double [HexQ.episodes];
	}
```

```
/*sinartisi pou orizei to xrono ka8e epeisodiou*/
public static void addTimerPerEpisode(int episode, double time){
	timerPerEpisode[episode]=time;
}


public static void main(String[] args) {

	try {
		wr2=new PrintWriter("Avg_Steps_Per_Episode.txt");
	} catch (FileNotFoundException e) {
		// TODO Auto-generated catch block
		e.printStackTrace();
	}
	try {
		wr3=new PrintWriter("Avg_Time_Per_Episode.txt");
	} catch (FileNotFoundException e) {
		// TODO Auto-generated catch block
		e.printStackTrace();
	}
	try {
		wr4=new PrintWriter("Avg_Reward_Per_Episode.txt");
	} catch (FileNotFoundException e) {
		// TODO Auto-generated catch block
		e.printStackTrace();
	}
	// TODO Auto-generated method stub
	HexQ.chooseTrials();
	HexQ.chooseEpisodes();
	HexQ.chooseDecay();
	HexQ.chooseDiscFactor();
	createMatrixes();

	HexQ.chooseEpsilon();
	HexQ.chooseLearningRate();
	HexQ.insertFilename();
	SA_Gridworld.makeGridworld(HexQ.filename);
	for ( i=0;i<HexQ.trials;i++){
		System.out.println("Trial "+i+"\n");
		HexQ.main(null);
		System.out.println("--------------------");

	}

	for(int i =0;i<HexQ.episodes;i++){
		int sum=0;
		for (int j=0;j<HexQ.trials;j++){
			sum+=matrix[i][j];
		}
		AverageStepsPerEpisode[i]=(double)sum/HexQ.trials;
		wr2.println(AverageStepsPerEpisode[i]);
	}
	for(int i =0;i<HexQ.episodes;i++){
		double sum2=0;
		for (int j=0;j<HexQ.trials;j++){
			sum2+=matrix2[i][j];
```

```
                }
                AverageTimePerEpisode[i]=(double)sum2/HexQ.trials;
                wr3.println(AverageTimePerEpisode[i]);
            }
            for(int i =0;i<HexQ.episodes;i++){
                double sum2=0;
                for (int j=0;j<HexQ.trials;j++){
                    sum2+=matrix3[i][j];
                }
                AverageRewardPerEpisode[i]=(double)sum2/HexQ.trials;
                wr4.println(AverageRewardPerEpisode[i]);
            }


        wr2.close();wr3.close();wr4.close();
        }

}
```

ΚΛΑΣΕΙΣ ΠΟΥ ΧΡΗΣΙΜΟΠΟΙΗΘΗΚΑΝ ΑΠΟ ΤΗΝ ΕΡΓΑΣΙΑ ΤΟΥ Ι.ΛΑΜΠΡΟΥ.

```
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;

/**
 * klasi pou antiprosopevi ena geniko gridworld
 *
 * @author Giannis
 *
 */
public class SA_Gridworld {

	public static int size_x; // grammes tou grid
	public static int size_y; // stiles tou grid
	public static ArrayList<Wall> wall = new ArrayList<Wall>();;
	public static int numOfStartPosition; // kai mesa sto ta3i sto taxi
problem
	public static int numOfGoals;
	public static int states;
	public static int actions;
	private static int[] pasLoc; // passenger location

	private static int[] destLoc; // destination location

	// rewards gia kathe action
	public static int wall_hit;
	public static int wrong_put;
	public static int wrong_get;
	public static int success;
	public static int simple_move;

	// statheres gia actions
	public static final int north = 0;
	public static final int south = 1;
	public static final int east = 2;
	public static final int west = 3;
	public static final int pickup = 4;
	public static final int putdown = 5;

	public static void setStates() {

		states = size_x * size_y * numOfStartPosition * numOfGoals;
	}

	/**
	 * methodos i opia epistrefi ton arithmo ton grammwn (x) tou grid
	 *
	 * @return the size_x
	 */
	public static int getSize_x() {
```

```
            return size_x;
      }

      /**
       * methodos i opia epistrefi ton arithmo ton stilon (y) tou grid
       *
       * @return the size_y
       */
      public static int getSize_y() {
            return size_y;
      }

      /**
       * methodos i opoia epistrefi ton pinaka me tis sintetagmenes twn
wall
       *
       * @return the wall
       */
      public ArrayList<Wall> getWall() {
            return wall;
      }

      /**
       * /** methodos i opoia epistrefi ton arithmo ton arxikon thesewn
pou
       * iparxoun sto grid
       *
       * @return the numOfStartPosition
       */
      public static int getNumOfStartPosition() {
            return numOfStartPosition;
      }

      /**
       * methodos i opoia epistrefi ton arithmon ton telikon thesewn pou
iparxoun
       * sto grid
       *
       * @return the numOfGoals
       */
      public static int getNumOfGoals() {
            return numOfGoals;
      }

      /**
       * methodos i opoia dimiourgi ena wall meta3i ton dio cells
       *
       * @param cell1
       * @param cell2
       */
      public static void setWall(int cell1, int cell2) {
            if (((Math.abs(cell1 - cell2) == 1) || ((Math.abs(cell1 -
cell2) == getSize_y())))
                        && (cell1 >= 0)
                        && (cell1 < ((getSize_x() * getSize_y())))
```

```
                        && (cell2 >= 0) && (cell2 < ((getSize_x() *
getSize_y()))))) {
                  Wall temp = new Wall(cell1, cell2);
                  wall.add(temp);
            } else {
                  System.err
                              .println("You cannot enter a wall between
these cells\nExiting..");
                  System.exit(-1);
            }

      }

      /**
       * methodos i opoia filaei ti thesi tou passenger
       *
       * @param i
       * @param location
       */
      public static void setPasLoc(int i, int location) {
            pasLoc[i] = location;
      }

      /**
       * methodos i opoia epistrefi tin thesi tou passenger simfwna me
tin
       * metavliti i
       *
       * @param i
       * @return
       */
      public static int getPasLoc(int i) {
            return pasLoc[i];
      }

      /**
       * methodos i opoia anatheti filaei to destination
       *
       * @param i
       * @param location
       */
      public static void setDestLoc(int i, int location) {
            destLoc[i] = location;
      }

      /**
       * methodos i opoia epistrefi to destination
       *
       * @param i
       * @return
       */
      public static int getDestLoc(int i) {
            return destLoc[i];
      }

      /**
```

```
	 * methodos i opoia elegxi an iparxi wall stin katefthinsi opou
theli na
	 * metakinithi o agent epistrefi true an iparxi kai false an den
iparxi
	 *
	 * @param cur_pos
	 * @param next_pos
	 * @return
	 */
	public static boolean checkForWall(int cur_pos, int next_pos) {
		for (int i = 0; i < wall.size(); i++) {
			if (((cur_pos == wall.get(i).cell1) && (next_pos ==
wall.get(i).cell2))
				|| ((cur_pos == wall.get(i).cell2) &&
(next_pos == wall
					.get(i).cell1)))
				return true;
		}
		return false;
	}

	/**
	 * methodos i opoia dimiourga to gridworld
	 * @param filename
	 */
	public static void makeGridworld(String filename) {
		BufferedReader in = null;
		String[] splitline;
		//System.out.println(filename);
		try {
			in = new BufferedReader(new FileReader(filename));
		} catch (FileNotFoundException e) {
			e.printStackTrace();
		}
		String line = null;

		// grammes tou gridworld
		try {
			line = in.readLine();
		} catch (IOException e) {
			e.printStackTrace();
		}
		splitline = line.split("=");
		size_x = Integer.parseInt(splitline[1]);

		// stiles tou gridworld
		try {
			line = in.readLine();
		} catch (IOException e) {
			e.printStackTrace();
		}
		splitline = line.split("=");
		size_y = Integer.parseInt(splitline[1]);

		// actions
		try {
```

```
		line = in.readLine();
	} catch (IOException e) {
		e.printStackTrace();
	}
	splitline = line.split("=");
	actions = Integer.parseInt(splitline[1]);

	// number of start position
	try {
		line = in.readLine();
	} catch (IOException e) {
		e.printStackTrace();
	}
	splitline = line.split("=");
	numOfStartPosition = Integer.parseInt(splitline[1]);
	pasLoc = new int[numOfStartPosition];

	// start positions
	try {
		line = in.readLine();
	} catch (IOException e) {
		e.printStackTrace();
	}
	splitline = line.split("=");
	String[] temp = splitline[1].split(",");
	for (int i = 0; i < temp.length; i++)
		setPasLoc(i, Integer.parseInt(temp[i]));


	// number of destinations
	try {
		line = in.readLine();
	} catch (IOException e) {
		e.printStackTrace();
	}
	splitline = line.split("=");
	numOfGoals = Integer.parseInt(splitline[1]);
	destLoc = new int[numOfGoals];

	// destinations
	try {
		line = in.readLine();
	} catch (IOException e) {
		e.printStackTrace();
	}
	splitline = line.split("=");
	temp = splitline[1].split(",");
	for (int i = 0; i < temp.length; i++)
		setDestLoc(i, Integer.parseInt(temp[i]));


	// walls
	try {
		line = in.readLine();
	} catch (IOException e) {
		e.printStackTrace();
```

```
            }
            splitline = line.split("=");
            temp = splitline[1].split(",");
            String[] tmp;
            for (int i = 0; i < temp.length; i++) {
                  tmp = temp[i].split("-");
                  setWall(Integer.parseInt(tmp[0]),
Integer.parseInt(tmp[1]));
            }

            // reward for wall hit
            try {
                  line = in.readLine();
            } catch (IOException e) {
                  e.printStackTrace();
            }
            splitline = line.split("=");
            wall_hit = Integer.parseInt(splitline[1]);

            // reward for simple move
            try {
                  line = in.readLine();
            } catch (IOException e) {
                  e.printStackTrace();
            }
            splitline = line.split("=");
            simple_move = Integer.parseInt(splitline[1]);

            // reward for wrong get
            try {
                  line = in.readLine();
            } catch (IOException e) {
                  e.printStackTrace();
            }
            splitline = line.split("=");
            wrong_get = Integer.parseInt(splitline[1]);

            // reward for wrong put
            try {
                  line = in.readLine();
            } catch (IOException e) {
                  e.printStackTrace();
            }
            splitline = line.split("=");
            wrong_put = Integer.parseInt(splitline[1]);

            // reward for succes
            try {
                  line = in.readLine();
            } catch (IOException e) {
                  e.printStackTrace();
            }
            splitline = line.split("=");
            success = Integer.parseInt(splitline[1]);

            try {
```

```
                line = in.readLine();
            } catch (IOException e) {
                e.printStackTrace();
            }

            setStates();
            //System.out.println(filename);
        }

        public static void main(String[] args) {
            SA_Gridworld.makeGridworld("singleSmall.txt");
        }
}

/**
 * klasi i opoia antiprosopevi ta state variables tou episodiou gia
single agent
 */
public class State_Variables {

        // Constructor 1
        State_Variables() {
            super();
        }

        // Constructor 2
        State_Variables(int size_x, int size_y, int numOfStartPosition,
        int numOfGoals) {
            int tmp = size_x * size_y;
            this.taxi = SA_Taxi_Problem.r.nextInt(tmp);
            this.passenger =
SA_Taxi_Problem.r.nextInt(numOfStartPosition);
            this.destination = SA_Taxi_Problem.r.nextInt(numOfGoals);
        }

        // Constructor 3
        State_Variables(int taxi, int passenger, int destination) {
            this.taxi = taxi;
            this.passenger = passenger;
            this.destination = destination;
        }

        // Constructor 4
        State_Variables(State_Variables s) {
            this.taxi = s.taxi;
            this.passenger = s.passenger;
            this.destination = s.destination;
        }

        int taxi;
        int passenger;
        int destination;
}
```