Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Τεχνητή Νοημοσύνη

11 Φεβρουαρίου 2012

Διονύσης Ζήνδρος (03106601)
dionyziz@gmail.com

Ερώτηση 1

Στο πρόβλημα ασχοληθήκαμε με την εύρεση λύσης σε χώρο καταστάσεων. Συγκεκριμένα υλοποιήθηκε ο αλγόριθμος A* έτσι ώστε δύο ρομπότ να βρουν το δρόμο τους προς ένα στόχο σε ένα χάρτη 2 διαστάσεων (κάτοψη) από μία δεδομένη θέση προς κάποιο δεδομένο στόχο.

Καθώς τα ρομπότ έχουν πλήρη γνώση όλου του περιβάλλοντός τους και των κινήσεων του άλλου ρομπότ και το περιβάλλον είναι στατικό, το πρόβλημα μπορεί να λυθεί χωρίς να χρειαστεί τα ρομπότ να αλλάζουν τις αποφάσεις τους κατά τη διάρκεια της κίνησής τους. Δηλαδή η διαδρομή μπορεί να σκιαγραφηθεί εξ' αρχής και να σχεδιαστεί η κίνηση του καθενός πριν να ξεκινήσει.

Για την επίλυση του προβλήματος χρησιμοποιήθηκε ο αλγόριθμος A*, μία παραλλαγή του αλγόριθμου του Dijkstra με την προσθήκη της ευριστικής συνάρτησης Manhattan για εκτίμηση αποστάσεων. Η εκτίμηση αποστάσεων γίνεται έτσι ώστε η διάσχιση του χώρου καταστάσεων να είναι συντομότερη. Η εύρεση λύσης δηλαδή επιτυγχάνεται πιο γρήγορα και με λιγότερες αναζητήσεις.

Είναι φανερό ότι, καθώς τα δύο ρομπότ κινούνται γνωρίζοντας το ένα για το άλλο, το πρόβλημα μπορεί να λυθεί τρέχοντας δύο ανεξάρτητες φορές τον αλγόριθμο A*, μία φορά για κάθε ρομπότ. Στη συνέχεια, αφότου βρεθεί το πλάνο κίνησης, τα ρομπότ μπορούν να φροντίσουν το ένα από τα δύο (π.χ. πάντα το A) να περιμένει (δηλαδή για μία χρονική στιγμή να μείνει ακίνητο) σε περίπτωση που το σχέδιό τους περιλαμβάνει να βρίσκονται στην ίδια θέση σε κάποια ορισμένη χρονική στιγμή. Επιπλέον, το ρομπότ που θα φτάσει δεύτερο στο στόχο μπορεί να μείνει σε κουτάκι που είναι γειτονικό του στόχου, αφού αυτό θα είναι άλλωστε σίγουρα μέρος της διαδρομής του.

Για την υλοποίηση επιλέχθηκε η γλώσσα C++ λόγω του ότι προσφέρει χρήσιμες έτοιμες βιβλιοθήκες δομών δεδομένων που βοηθάνε στην επίλυση του προβλήματος. Συγκεκριμένα χρησιμοποιήθηκε η STL με τις δομές δεδομένων set, vector, list. Επιπλέον, το πρόγραμμα χωρίστηκε σε 3 αρχεία: robots.{cpp,h}, astar.{cpp,h} και visualizer.{cpp,h}. Το αρχείο robots αναλαμβάνει την ανάγνωση των δεδομένων εισόδου και λειτουργεί ως χειριστής για ολόκληρο το πρόγραμμα. Το astar είναι υπεύθυνο για τη λογική της αναζήτησης σε χώρο καταστάσεων χρησιμοποιώντας τον αλγόριθμο A*. Το visualizer αναλαμβάνει την οπτικοποίηση του αποτελέσματος σε εικόνα μορφής PNG. Για τη μεταγλώττιση χρησιμοποιήθηκε ο gcc και μαζί με τον πηγαίο κώδικα υπάρχει και εύχρηστο αρχείο Makefile.

Η οπτικοποίηση έγινε χρησιμοποιώντας τη C βιβλιοθήκη γραφικών cairo. Η βιβλιοθήκη επιλέχθηκε λόγω του καθαρού και απλού API. Το πρόγραμμα παράγει ένα αρχείο PNG το οποίο αναπαριστά οπτικά το χάρτη όπου φαίνονται καθαρά ο ελεύθερος χώρος κίνησης , τα εμπόδια, καθώς και τις κινήσεις που έκαναν τα δύο robot στο χάρτη, από την αρχική τους θέση έως το στόχο τους (robots.png).

Εκτός από την οπτική αναπαράσταση, το πρόγραμμα παράγει επίσης και αναπαράσταση κειμένου του αποτελέσματος που περιλαμβάνει τη διαδρομή που βρέθηκε για κάθε robot καθώς και οδηγίες για την αποφυγή συγκρούσεων.

Ερώτηση 2

Η δομή δεδομένων που χρησιμοποιήθηκε για αναζήτηση στο χώρο καταστάσεων ήταν η εξής κλάση της C++:

```
struct Edge {
    public:
        int parent;
        Point from;
        Point to;

        // approximation of how far the "to" end of Edge is from the target
        int heuristic;

        // actual distance between the "to" end of the Edge and the source
        int distance;
};
```

Η κλάση αυτή αντιπροσωπεύει μία πιθανή μετάβαση από ένα κουτάκι σε ένα άλλο· περιλαμβάνει το κουτάκι προέλευσης, το κουτάκι προορισμού, την ευριστική εκτίμηση μέχρι το στόχο, καθώς και την πραγματική απόσταση που έχουμε διανύσει από την πηγή.

Ερώτηση 3

Η υλοποίηση του αλγορίθμου A* γίνεται μέσω δύο συναρτήσεων, της συνάρτησης aStar και της συνάρτησης enqueue. Η συνάρτηση aStar περιέχει τη βασική λογική υλοποίησης του αλγορίθμου. Εκεί, ο αλγόριθμος ξεκινάει προσθέτοντας σε μία ουρά τις συνδέσεις μεταξύ θέσεων στο χάρτη που μπορεί να χρησιμοποιήσει. Ο αλγόριθμος στη συνέχεια χρησιμοποιεί την ουρά ως ουρά προτεραιότητας ταξινομώντας τα υποψήφια στοιχεία με βάση μία συνάρτηση που αξιολογεί κάθε ακμή σύμφωνα με μία εκτιμώμενη απόσταση από το στόχο. Η συνάρτηση aStar αφορά μόνο μία συγκεκριμένη πηγή και ένα συγκεκριμένο στόχο, συνεπώς καλείται μια φορά για κάθε ρομπότ. Η συνάρτηση επίσης περιέχει τη λογική πιθανών μεταβάσεων από κουτάκι σε κουτάκι του χάρτη. Η συνάρτηση enqueue χρησιμοποιείται βοηθητικά εντός της συνάρτησης aStar για να προσθέσει μία νέα πιθανή επιλογή στην ουρά προτεραιότητας. Όλες οι συναρτήσεις είναι κατασκευασμένες να μην χρησιμοποιούν global μεταβλητές και με διαφάνεια αναφοράς ώστε να είναι επαναχρησιμοποιήσιμες σε μορφή βιβλιοθήκης.

Ερώτηση 4

Ως admissible heuristic χρησιμοποιήθηκε η συνάρτηση απόστασης Manhattan. Ο λόγος που είναι admissible είναι διότι ένα robot χρειάζετα τόσα βήματα όσα δείχνει η απόστασης Manhattan για να φτάσει στο στόχο στην καλύτερη περίπτωση, αν δεν συναντήσει εμπόδια. Διαφορετικά χρειάζεται περισσότερα βήματα. Ως non-admissible heuristic χρησιμοποιήθηκε η συνάρτηση $x^2 + y^2$. Αυτή η συνάρτηση είναι non-admissible heuristic. Ο

λόγος είναι ότι υπερεκτιμά την απόσταση προς το στόχο. Για παράδειγμα, αν δεν υπάρχουν καθόλου εμπόδια ανάμεσα στη θέση που βρίσκεται το robot και στο στόχο, τότε ο υπερεκτιμητής θα δώσει ως αποτέλεσμα μεγαλύτερο απ' ό,τι η πραγματική απόσταση που θα διένυε η βέλτιστη λύση. Γι' αυτό το λόγο, ενδέχεται ο υπερεκτιμητής να οδηγηθεί σε υποβέλτιστα αποτελέσματα, αν και σε λιγότερο πλήθος βημάτων.

Ερώτηση 5

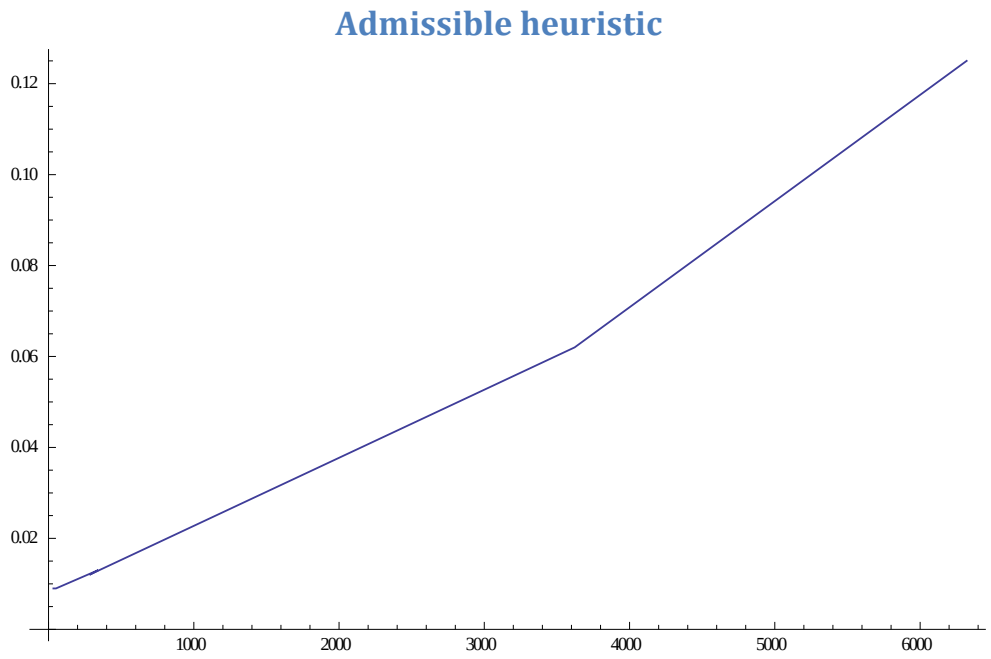
Ο αλγόριθμος που μπορεί να χρησιμοποιηθεί για την αποφυγή συγκρούσεων μπορεί να τρέξει μετά την εφαρμογή του A* και για τα δύο robot και να κάνει το robot A (που επιλέγεται ώστε να παραχωρεί προτεραιότητα) απλώς να περιμένει αν το robot B βρίσκεται στο δρόμο του μία δεδομένη χρονική στιγμή και είναι ο ακόλουθος:

```
Resolve-Collision( PathA, PathB ):
        NewPathA = []
        j ← 0
        for t ← 0 to |PathA| - 2:
                if PathB[ i ] = PathA[ i ]:
                        NewPathA[ j ] ← WAIT
                else:
                        NewPathA[ j ] ← PathA[ i ]
        return ( NewPathA, PathB )
```

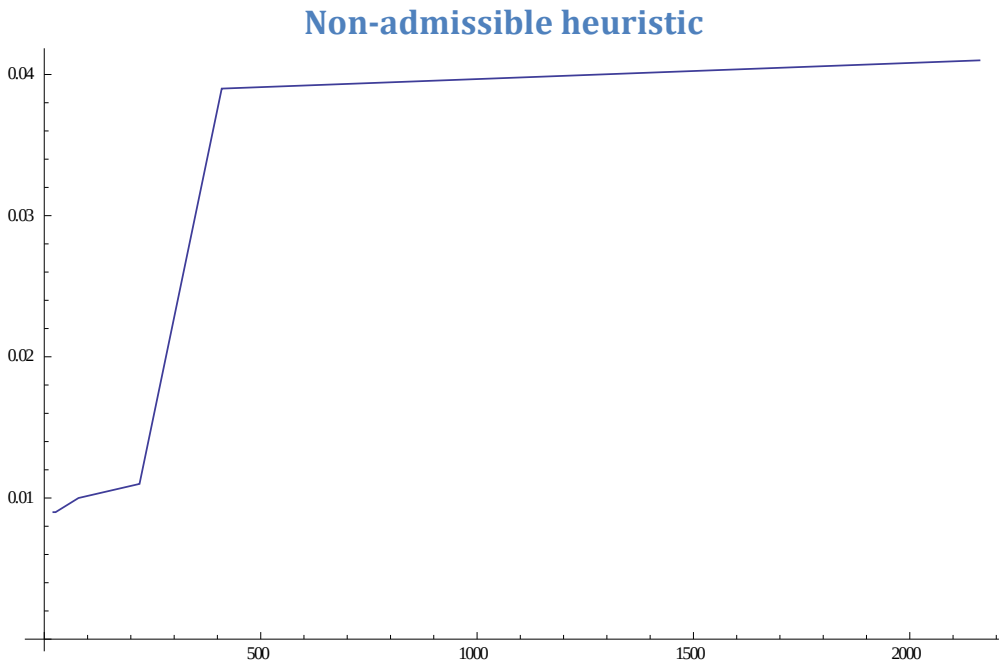
Η υλοποίηση της WAIT μπορεί να γίνει βάζοντας το ρομπότ να πηγαινοέρχεται ανάμεσα στα δύο τετραγωνάκια που επισκέφθηκε πιο πρόσφατα για ένα γύρο αν είμαστε υποχρεωμένοι να κινούμαστε. Στη συνέχεια, όποιο από τα ρομπότ A ή B φτάνει τελευταίο (δηλαδή έχει λόγου χάρη |PathA| < |PathB|), μπορεί να αφαιρέσει το τελευταίο βήμα από τη διαδρομή του ώστε να καταλήξει σε κουτάκι γειτονικό του στόχου.

Ερώτηση 6

Παρατίθενται τα στατιστικά αποτελέσματα της εκτέλεσης.



Χρόνος εκτέλεσης συναρτήσει του αριθμού των κόμβων



Χρόνος εκτέλεσης συναρτήσει του αριθμού των κόμβων

Και στις δύο περιπτώσεις ο χρόνος εκτέλεσης είναι αμελητέος για τις μικρές δοκιμαστικές περιπτώσεις και πολύ μικρός για τις μεγαλύτερες. Για την μέτρηση χρησιμοποιήθηκε η κλήση time του UNIX, χωρίς να συμπεριλαμβάνονται οι χρόνοι εκτύπωσης των αποτελεσμάτων και οπτικοποίησης. Ως δοκιμαστικά αρχεία χρησιμοποιήθηκαν τα δοκιμαστικά αρχεία ελέγχου των Γρηγόρη Λύρα και Βασίλη Γερακάρη που συμπεριλαμβάνονται στο tarball του πηγαίου κώδικα.

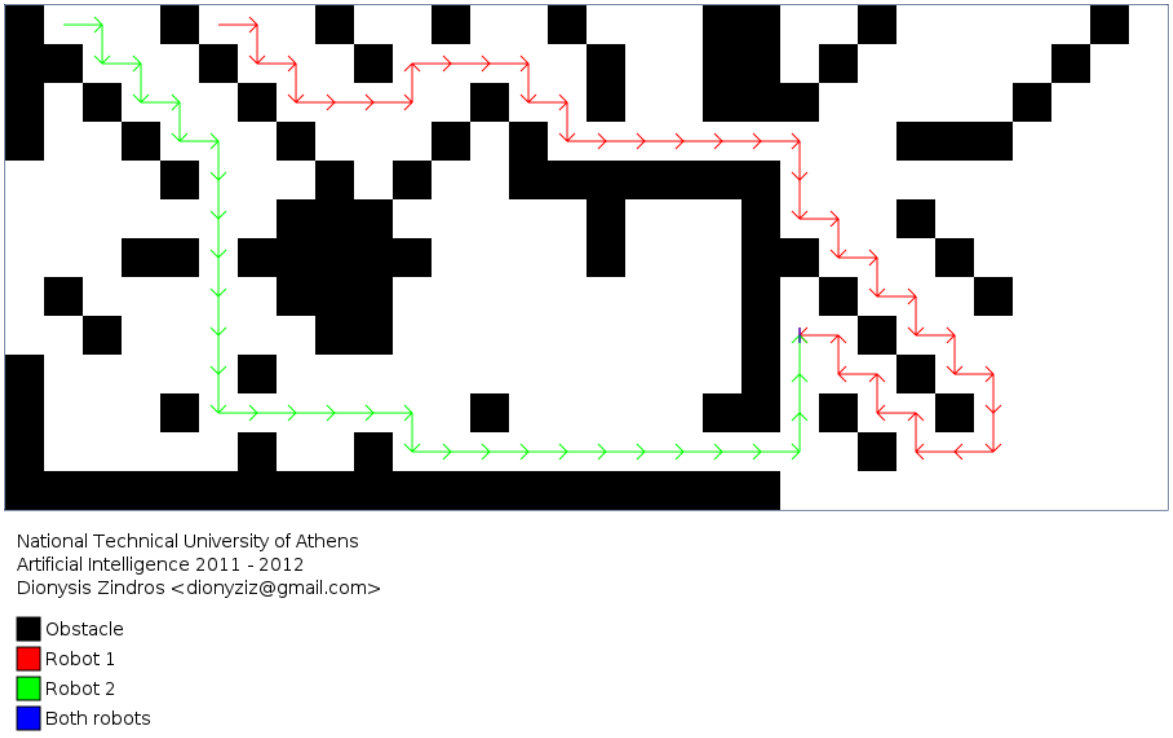
Ερώτηση 7

Παρακάτω φαίνεται το text output του προγράμματος σε περίπτωση εντοπισμού σύγκρουσης για ένα παράδειγμα κίνησης:

```
Robot A path:

(1, 2), (1, 3), (1, 4), (1, 5), (2, 5), (3, 5), (4, 5), (5, 5), (6, 5), (7, 5)

Robot B path:

(4, 1), (3, 1), (3, 2), (3, 3), (3, 4), (3, 5), (4, 5), (5, 5), (6, 5), (7, 5)

First potential collision at ( 3, 5 ).

Avoiding collision by making Robot A wait.
```

Ερώτηση 8

Παρακάτω φαίνονται τα βήματα το κάθε robot για ένα από τα input files, όπως παράγονται από το πρόγραμμα σε μορφή εικόνας:



Ερώτηση 9

Ο πηγαίος κώδικας του προγράμματος υπάρχει στο tarball της εργασίας. Ενημερωμένη έκδοση υπάρχει στο https://github.com/dionyziz/ntua-ai, μαζί με το πλήρες ιστορικό ανάπτυξης του προγράμματος και τα δοκιμαστικά αρχεία.