Τεχνητή Νοημοσύνη
2ο Προγραμματιστικό Θέμα
Αναφορά Εργασίας
Ακ. Έτος 2018-2019

Μέλη ομάδας (αλφαβητικά)

Εμμανουήλ Παναγιώτου (03115079) – el15079@mail.ntua.gr

Μάριος Παπαχρήστου (03115101) – papachristoumarios@gmail.com

"It's going to be interesting to see how society deals with artificial intelligence, but it will definitely be cool". –Colin Angle

1 Σκοπός Εργασίας

Σε αυτή την εργασία υλοποιούμε το βασικό κορμό μιας ευφυούς υπηρεσίας εξυπηρέτησης πελατών ταξί. Συγκεκριμένα,θεωρούμε ότι υπάρχει ένας πελάτης που βρίσκεται σε μια ορισμένη τοποθεσία, ο οποίος διαθέτει κινητό τηλέφωνο με GPS και επιθυμεί να καλέσει ένα ταξί. Η υπηρεσία διαθέτει μια βάση δεδομένων με όλα τα διαθέσιμα ταξί και τη γεωγραφική θέση στην οποία βρίσκονται κάθε χρονική στιγμή, η οποία θεωρητικά θα πρέπει να ανανεώνεται συνεχώς. Η υπηρεσία θα πρέπει να εντοπίζει και να ειδοποιεί το ταξί που μπορεί να μεταβεί πιο γρήγορα στη θέση του πελάτη ώστε να τον εξυπηρετήσει. Ο χάρτης δίνεται απο δειγματοληπτημένα σημεία οδών από την πόλη της Αθήνας και ο τρόπος εύρεσης της βέλτιστης επιλογής γίνεται με τη χρήση του αλγορίθμου A* με τη δυνατότητα επιλογής πολλών εναλλακτικών διαδρομών (similar ETA). Στη συνέχεια ο πελάτης πρέπει να μετακινηθεί βέλτιστα στον προορισμό του.

Στη συγκεκριμένη περίπτωση θα παρέχεται ένας πυρήνας σε Java με τις κύριες λειτουργίες του αλγορίθμου, ενώ όλη η γνώση των πρακτόρων προέρχεται από κατηγορήματα σε γλώσα Prolog. Για την διασύνδεση Java και Prolog έχει χρησιμοποιηθεί η βιβλιοθήκη JIProlog.

2 Εκτέλεση εργασίας

Στην εργασία παρέχεται ένα Makefile για την εκτέλεση των σεναρίων. Για να εκτελέσουμε όλη την εργασία χρησιμοποιούμε την `make all`. Η κύρια κλάση βρίσκεται στο `Graph.java`. Αν επιθυμείτε να τρέξετε τον αλγόριθμο για πολλαπλά tolerances χρησιμοποιείστε την `make evaluate`.

3 Οργάνωση και Διάρθρωση Γνώσης για τον Κόσμο

Θεωρώντας ότι τα ταξί και οι πελάτες ζουν σε ένα κόσμο με κάποιες ιδιότητες, μπορούμε να ορίσουμε μια σειρά από κατηγορήματα σε Prolog που αναπαριστούν όλα αυτά τα σενάρια. Συγκεκριμένα, θεωρούμε ότι ο χάρτης αναπαρήσταται από ένα κατευθυνόμενο γράφημα $G(V, E)$ όπου το κατηγόρημα `next/4` αναφέρεται σε ακμή του γραφήματος, το κατηγόρημα `lines/20` σε οδούς στο δρόμο, το κατηγόρημα `taxis/10` στα ταξί, το κατηγόρημα `client/9` στους πελάτες και το κατηγόρημα `traffic/5` στην κίνηση της εκάστοτε οδού. Τα κατηγορήματα αυτά κατασκευάζονται από τα αρχεία CSV που μας παρέχονται και την κλάση `FactGenerator`. Παραθέτουμε μερικά παραδείγματα

```
traffic(5203817,mask,[09:00-11:00=high,13:00-15:00=medium,17:00-19:00=high], mask,
    mask).
client(0,23.733912,37.975687,23.772518,38.012301,20:00,3,greek,1).
taxis(23.789114,38.032908,130,no,1-4,[greek,english],8.3,yes,subcompact,mask).
```

όπου έχουμε χρησιμοποιήσει το άτομο `mask` για τους χαρακτήρες που δεν είναι ASCII ή δεν υπάρχει πληροφορία.

Στη συνέχεια ορίζουμε κατηγορήματα που αφορούν μεταβάσεις και λοιπές πληροφορίες για τον κόσμο στο αρχείο `world.pl` όπως φαίνεται παρακάτω και στο οποίο κάνουμε consult όλη τη βάση γνώσης.

```
% Consults
:- [
    lines,
    client,
    taxis,
    next,
    traffic
    ].

% Node belongs to line L
belongsTo(X, Y, L) :- nextWithLine(X, Y, _, _, L).

% Line is directed same as nodes
directed(L) :-
  lineDirection(L, yes).

% True when line is directed opposite
oppositeDirected(L) :-
  lineDirection(L, -1).

% True when line is undirected
undirected(L) :-
  lineDirection(L, no).

% Next brings the adjacent nodes of a node
next(X, Y, U, V) :- nextWithLine(X, Y, U, V, _).

% Is true iff there is a path from X, Y to U, V
canMoveFromTo(X, Y, U, V) :-
  next(X, Y, P, Q),
  next(P, Q, U, V).

% Get adjacent nodes avoiding certain types of traffic
% Example nextAvoidingTraffic(23.7611292,37.9863578, U, V, 0, [high, medium])
nextAvoidingTraffic(X, Y, U, V, ClientId, TrafficToAvoid) :-
  nextWithLine(X, Y, U, V, L),
  client(ClientId, _, _, _, _, H:M, _, _, _),
  traffic(L,_,Traffics,_, _),
  avoidsTrafficTypes(Traffics, H:M, TrafficToAvoid).
```

```
% Get all goals
goal(ClientId, TaxiId) :-
  client(ClientId, _, _, _, _, _, Capacity, ClientLanguages, Luggage),
  taxis(X, Y, TaxiId, yes, Min-Max, TaxiLanguages, _, _, Type, _),
  between(Min, Max, Capacity),
  fitsLuggage(Type, Luggage),
  ssubset(ClientLanguages, TaxiLanguages).

% Get all goals for a client
goals(ClientId, Taxis) :-
  findall(T, goal(ClientId, T), Taxis).

% Check if sets are not disjoint
doesIntersect(X,Y) :-
  intersection(X,Y,Z),
  dif(Z,[]).

% Checks if Source is ssubset of Target
ssubset(Source, Target) :-
  (
  Source = [_ | _], Target = [_ | _] -> doesIntersect(Source, Target);
  Target = [_ | _] -> member(Source, Target);
  Source = Target
  ).

% Check is H3:M3 is in interval
betweenHours(H1:M1, H2:M2, H3:M3) :-
  X1 is 60 * H1 + M1,
  X2 is 60 * H2 + M2,
  X3 is 60 * H3 + M3,
  between(X1, X2, X3).

% Avoid Traffic
hasTraffic(Traffics, H:M, Type) :-
  member(H1:M1-H2:M2-Type, Traffics),
  betweenHours(H1:M1, H2:M2, H:M).

% Avoid certain types of traffic
avoidsTrafficTypes(_, _, []).
avoidsTrafficTypes(_, _, mask).
avoidsTrafficTypes(Traffics, H:M, [Type | Rest]) :-
  \+hasTraffic(Traffics, H:M, Type),
  avoidsTrafficTypes(Traffics, H:M, Rest).

% Fits luggage if any
fitsLuggage(Type, Luggage) :-
  (
    Luggage = 1 -> member(Type, [large, subcompact]);
    true
  ).


  % Distance predicate
  haversine(Lon1, Lat1, Lon2, Lat2, H) :-
    Theta is Lon1 - Lon2,
    raddians(Lat1, RLat1),
    raddians(Lat2, RLat2),
    raddians(Theta, Rtheta),
    S1 is sin(RLat1),
    S2 is sin(RLat2),
```

```
    C1 is cos(RLat1),
    C2 is cos(RLat2),
    CT is cos(RLat2),
    Distance is (S1 * S2) + (C1 * C2 * CT),
    ACos is acos(Distance),
    deggrees(ACos, D),
    H is D * 111.18957696.


% Get taxi rating
rating(I, R) :-
  taxis(_, _, I, _, _, _, R, _, _, _).

% Default getters
getPoint(U, V) :- next(U, V, _, _); next(_, _, U, V).
getClient(I, X, Y, U, V) :- client(I, X, Y, U, V,_, _, _, _).
getTaxi(I, U, V) :-   taxis(U, V, I, _, _, _, _, _, _, _).

% Degrees to radians
raddians(P, Q) :-
  Q is P * pi / 180.

% Radians to degrees
deggrees(P, Q) :-
  Q is P * 180 / pi.
```

Εδώ παρατηρούμε ότι μπορούμε να επιτελέσουμε διάφορες λειτουργίες, όπως να βρούμε τα υποψήφια ταξί στόχους, να βρούμε γείτονες κόμβους που η μετάβαση δεν έχει συνωστισμό μια συγκεκριμένη ώρα κτλ. Στη συνέχεια, δημιουργούμε έναν wrapper μεταξύ του `world.pl` και της Java λειτουργικότητας μέσω της βιβλιοθήκης JIProlog στην κλάση `Prolog.java`.

4 Αντικειμενοστραφής Σχεδιασμός

Η παρούσα εργασία αναπτύχθηκε σε γλώσα Java, με γνώμονα τον αντικειμενοστραφή σχεδιασμό. Χρησιμοποιήθηκαν κλάσεις για τα Ταξί (Taxi), τους Πελάτες (Client), τις Κορυφές (Node), τις ακμές (Edge), τους εκτιμητές για τον A* (Estimator) και τον συνολικό γράφο (Graph). Το parsing των κορυφών γίνεται μέσω της κλήσης των κατάλληλων μεθόδων του wrapper. Τέλος χρησιμοποιήθηκε μια κλάση (Visual) για την παραγωγή των αρχείων KML ώστε να γίνεται η απεικόνιση των αποτελεσμάτων στον χάρτη. Για τις δομές δεδομένων χρησιμοποιήθηκαν επίσης κλάσεις των standard βιβλιοθηκών της Java.

5 Δημιουργία Γραφήματος

Ο γράφος χτίζεται on-line με τη χρήση των κατάλληλων κατηγορημάτων που υπάρχουν στο `world.pl` για την λήψη των adjacent κόμβων ενός κόμβου u μέσω της κλήσης της μεθόδου `next`. Το μήκος κάθε ακμής $\{u, v\} \in E$ τίθεται ίσο με την γεωγραφική απόσταση σε km δηλαδή

$$w(u, v) = \text{geo}(u, v) = 111.18957696 \times \arccos\left[\sin\phi_u \sin\phi_v + \cos\phi_u \cos\phi_v \cos(\theta_u - \theta_v)\right]$$

6 Αλγόριθμος A*

6.1 Περιγραφή Αλγορίθμου

Για την υλοποίηση του A* χρησιμοποιήθηκαν δύο hashtables για να κρατήσουμε τις $f(v)$ και $g(v)$. Επίσης χρησιμοποιήθηκε ένα HashSet για τη διατήρηση του κλειστού συνόλου και μια ουρά προτεραιό-

τητας `PriorityQueue<Estimator>` στην οποία τηρούταν το μέτωπο αναζήτησης (ανοικτό σύνολο). Κάθε φορά, από το μέτωπο αναζήτησης αφαιρούνταν ο κόμβος με την ελάχιστη τιμή

$$f(v) = g(v) + h(v)$$

όπου $h(v)$ η ευριστική συνάρτηση. Αν ο v ήταν στόχος τότε η αναζήτηση τερμάτιζε. Σε κάθε άλλη περίπτωση ο αλγόριθμος έκανε relax στις κορυφές στις οποίες βελτιώνονταν το $g(v)$.

6.2 Ευριστική Συνάρτηση

Δεδομένης της γεωμετρίας της πόλης, η ευριστική συνάρτηση που διαλέξαμε είναι η Νόρμα Ταξί L_1 από τον κοντινότερο στόχο με τύπο

$$h(v) = \min_{g \in G} h_g(v)$$

όπου $G = \{g \in V \mid g \text{ στόχος}\}$ με

$$h_g(v) = \lambda \text{geo}(v, g) \qquad \lambda \leq 1$$

Στη δική μας υλοποίηση θέσαμε $\lambda = 0.9$. Για να επιστρέψει σωστά αποτελέσματα ο A* θα πρέπει η ευριστική συνάρτηση να υποεκτιμά το πραγματικό κόστος προς το στόχο.

6.3 Αποθήκευση Διαδρομών

Για να τηρήσουμε τις διαδρομές χρησιμοποιούμε ένα `Hashtable<Node, ArrayList<Pair>>` και σε κάθε relaxation προσθέτουμε στη λίστα την τιμή που βρίσκουμε (η οποία μικραίνει συνεχώς). Ο πίνακας κατακερματισμού θα χρησιμεύσει στην εκτύπωση των συντομότερων διαδρομών. Ο κατευθυνόμενος υπολειμματικός γράφος H είναι DAG. Κάνοντας μια αναζήτηση από την κορυφή–στόχο στην αρχή μπορούμε να φτιάξουμε το γράφημα των ισοδύναμων βέλτιστων διαδρομών για τις τιμές των εκτιμήσεων της πραγματικής απόστασης του κάθε κόμβου σε κάθε χρονική στιγμή με τη βέλτιστη (συγκλίνουσα) εκτίμηση. Για τις ισοδύναμες διαδρομές έχουμε επιτρέψει κάποιο tolerance ϵ (της τάξης των 0.001km).

6.4 Αξιολόγηση Αλγορίθμου

Ο αλγόριθμος A* βρίσκει τη βέλτιστη λύση αν η ευριστική συνάρτηση $h(v)$ **υποεκτιμά το πραγματικό κόστος προς τον κοντινότερο στόχο**. Ο πίνακας κατακερματισμού για τους γονείς (parent) επιτρέπει σε κάθε έναν κόμβο να έχει γονείς εντός μιας ακρίβειας ϵ από τη βέλτιστη τιμή $g(v)$. Το συγκεκριμένο πρόβλημα πληροί τις προϋποθέσεις για να προτείνει ισοδύναμες ελάχιστες διαδρομές. Θέτοντας $\epsilon = 0$ λαμβάνουμε την πραγματική ελάχιστη διαδρομή. Οι ισοδύναμες διαδρομές βρίσκονται εντός κάποιας τιμής ακρίβειας από την πραγματική βέλτιστη. Στην πράξη, οι χάρτες προτείνουν "υπο"-βέλτιστες διαδρομές με παρόμοιο ETA.

7 Κατάταξη Ταξί

Πρώτη Κατάταξη Για να λάβουμε την πρώτη κατάταξη των ταξί τρέξαμε από κάθε ταξί A* προς τον πελάτη. Τα αποτελέσματα συγκεντρώθηκαν σε έναν πίνακα που είναι ταξινομημένος με βάση την απόσταση προς τον πελάτη σε αύξουσα σειρά. Τα πρώτα k αποτελέσματα εμφανίζονται στον πελάτη ως ακολούθως

```
Displaying Ranks
FIRST RANKING: of top 5 taxis according to distance criteria
1. Taxi: 100    Distance: 1.5149735932289947 km Rating: excellent
```

```
2. Taxi: 150    Distance: 2.210520017996553 km  Rating: good
3. Taxi: 120    Distance: 3.0738913585636634 km Rating: excellent
4. Taxi: 160    Distance: 3.71643311105736 km   Rating: average
5. Taxi: 170    Distance: 3.7512154487319935 km Rating: excellent
```

Δεύτερη Κατάταξη Για την δεύτερη κατάταξη των ταξί χρησιμοποιήθηκαν από τα πρώτα k ταξί της πρώτης κατάταξης μόνο αυτά που πληρούσαν το κατηγόρημα `goal/2`. Συγκεκριμένα σε αυτό το κατηγόρημα λαμβάνουμε τα διαθέσιμα ταξί τα οποία χωράνε τους πελάτες, δεν είναι κατειλλημένα εκείνη τη στιγμή (commute time), μιλάνε τη γλώσσα του πελάτη (κατηγόρημα `ssubset/2`), χωράνε τις αποσκευές (κατηγόρημα `fitsLuggage/2`). Από τα παραπάνω ταξί λαμβάνουμε επομένως τα 2 ταξί που πληρούν αυτές τις προϋποθέσεις

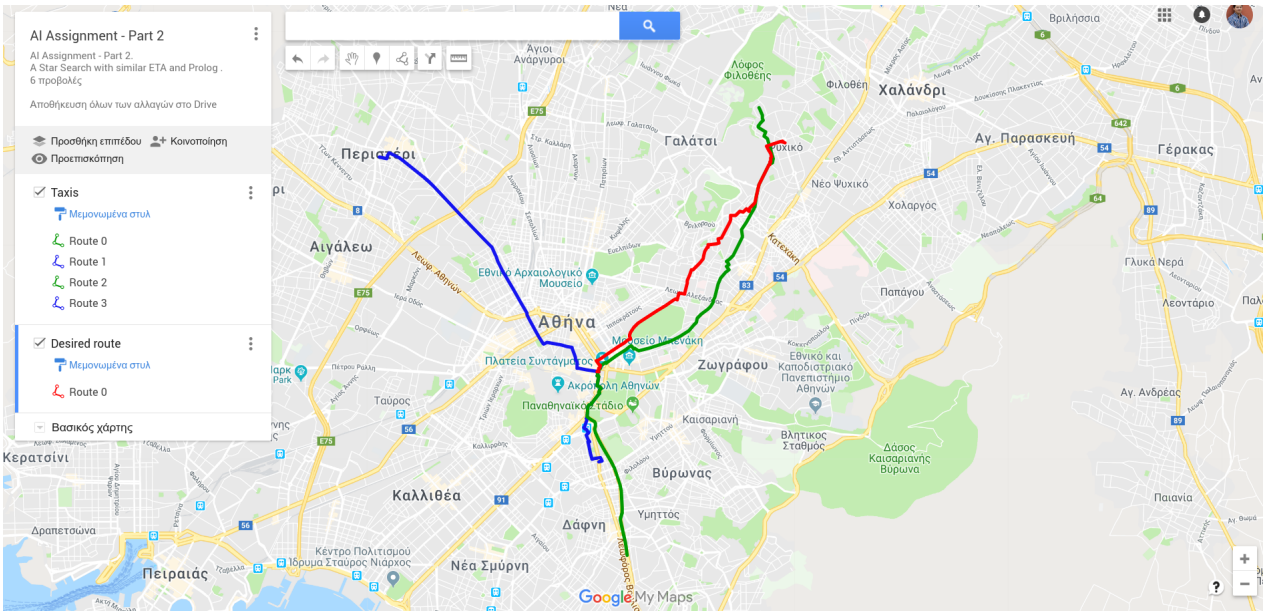
```
SECOND RANKING: The following taxis are available to serve you according to your
    language, commute time, capacity and availability
1. Taxi: 150
2. Taxi: 160
Enter a choice:
```

Τέλος, ο πελάτης καλείται να κάνει την επιλογή μεταξύ αυτών. Τέλος υπολογίζεται η συνολική διαδρομή για το ταξί (ταξί → πελάτης → προορισμός) και λαμβάνουμε το τελικό αποτέλεσμα (π.χ. για το ταξί 150)

```
Enter a choice: 150
Calculating distance to destination
Route length: 6.274812109190532 km
Total distance travelled by taxi 150 to destination: 8.485332127187085 km
```

8 Αποτελέσματα

Ο χάρτης με τα αποτελέσματα των διαδρομών βρίσκεται στο σύνδεσμο `https://drive.google.com/open?id=1mokuwgSJmQAvU3LJWCLfpjxm0EhD0h7q&usp=sharing`. Για το παράδειγμα το αποτέλεσμα των διαδρομών στο χάρτη (για μηδενικό tolerance) φαίνονται παρακάτω



Σχήμα 1: Αποτελέσματα διαδρομών για μηδενικό tolerance

Αναφορές

[1] Russell, Stuart J., and Peter Norvig. Artificial intelligence: a modern approach. Malaysia; Pearson Education Limited,, 2016.

[2] Στάμου Γ., Τεχνητή Νοημοσύνη, Διαφάνειες