ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΛΟΠΟΝΝΗΣΟΥ
ΣΧΟΛΗ ΟΙΚΟΝΟΜΙΑΣ, ΔΙΟΙΚΗΣΗΣ ΚΑΙ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

Θέματα διαχείρισης πληροφοριών και δεδομένων

Τελική Αναφορά

ΚΟΖΪΩΚΑΣ ΠΑΝΑΓΙΩΤΗΣ

[tst04009@uop.gr]

ΠΛΈΣΣΙΑΣ ΑΛΈΞΑΝΔΡΟΣ

[cst11068@uop.gr]

Εαρινό εξάμηνο 2014-2015

Περιεχόμενα

Ερώτημα 1

Περιγραφή αλγόριθμου Farthest

Επιλέγει ένα σημείο και στην συνέχεια βρίσκει ένα καινούργιο σημείο u που είναι πιο μακριά. Προσθέτει το u στο σετ των οροσήμων. Συνεχίζει τις συγκρίσεις και σε κάθε σύγκριση βρίσκει ένα καινούργιο σημείο που είναι πιο μακριά από το τρέχον σετ των οροσήμων και το προσθέτει στο σετ.

Ψευδοκώδικας:

```
for (int landmarkSelected = 0; landmarkSelected < LANDMARKCOUNT; landmarkSelected++) {
        for all_nodes(v,G) {
            if(v == s)
                dist[v] = 0;
        else
            dist[v] = MAXDOUBLE;
                PQ.insert(v,dist[v]);
        }
    while ( !PQ.empty() ) {
            u = PQ.del_min();
            if( dist[u] == MAXDOUBLE ) {
                    PQ.clear();
                    break;
            }
            for_all_adj_edges(e,u) {
                v = target(e);
                double c = dist[u] + cost[e];
                if ( c < dist[v] ) {
                    PQ.decrease_p(v,c); dist[v] = c;
                }
        }//Neighbour distance updation ends
    }//While the Priority Queue has vertices to be explored
  landmarks.append( u ); //Select the farthest node from s as landmark
  s = u; //Next Source for another landmark selection
} //Landmark Selection loop
```

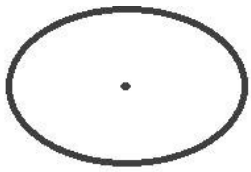
Η δομή landmarks είναι μία λίστα που περιέχει τα ορόσημα που επιλέγονται από την LANDMARKCOUNT;

Περιγραφή αλγόριθμου Planar

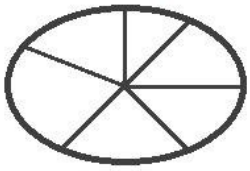
Βρίσκουμε ένα σημείο c όσο πιο κοντά στο κέντρο του χάρτη(εικόνα 1). Μετά χωρίζουμε τον χάρτη σε k κομμάτια (pie-slices)(εικόνα 2). Για κάθε κομμάτι επιλέγουμε ένα σημείο πιο μακριά από το κέντρο(εικόνα 3). Επιπλέον εάν ένα σημείο είναι κοντά στα σύνορα με διπλανό κομμάτι τα γειτονικά σημεία παραλείπονται έτσι ώστε να αποφύγουμε δύο σημεία να είναι κοντά.

Ψευδοκώδικας:

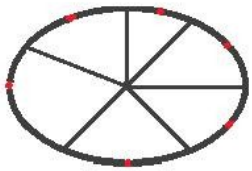
```
find.center();
divide.pie-slices();
for(i=0; i=pie-slices; i++)
{
        find.farthestSpot();
        (Εδώ πρέπει να γίνεται η σύγκριση για το εάν τα σημεία είναι κοντά)

}
```



Εικόνα 1



Εικόνα 2



Εικόνα 3

**Και για τις δύο υλοποιήσεις θα χρησιμοποιηθεί ο BFS αντί του Dijkstra για τον υπολογισμό των αποστάσεων

Στόχος

Να μεταφορτώσουμε το οδικό δίκτυο της Ελλάδας στη βάση PostgreSQL. Πρώτα να κατεβάσουμε από τον σύνδεσμο http://download.geofabrik.de/europe/greece-latest.osm.bz2 το OSM δίκτυο της Ελλάδας από το Geofabrik και να το φορτώσουμε στη βάση με το osm2pgrouting.

Διαδικασία εγκατάστασης του εργαλείου osm2pgrouting

Αρχικά, ανοίγοντας τον σύνδεσμο της εκφώνησης κατεβαίνει ένα αρχείο με όνομα greece-latest.osm.bz2 το οποίο το αποσυμπιέζουμε με την εντολή `bzip2 -dk greece-latest.osm.bz2` η οποία εξάγει το αρχείο greece-latest.osm και διατηρεί το αρχικό αρχείο.

Στην συνέχεια, κατεβάζουμε το osm2pgrouting από τον σύνδεσμο https://github.com/pgRouting/osm2pgrouting και ανοίγοντας τον κατάλογο osm2pgrouting-master εκτελούμε την εντολή: `cmake -H. -Bbuild` η οποία χρησιμοποιεί τους διακόπτες –H και –B οι οποίοι προσδιορίζουν την πηγή του cmake και τον δυαδικό κατάλογο κλήση του cmake, στην συνέχεια πηγαίνουμε στον κατάλογο build με την εντολή: `cd build/` και στον κατάλογο αυτόν εκτελούμε τις εντολές: `make` και `make install` οι οποίες κάνουν build και install αντίστοιχα με βάση το αρχείο Makefile, εδώ αξίζει να αναφερθεί πως χρειάστηκε να αλλάξουμε τον κατάλογο εγκατάστασης σε `/home/db3u01/osm2pgrouting` από `/usr/share/bin/osm2pgrouting` που ήταν το default ώστε να είναι εγκατεστημένο τοπικά στον χώρο μας μόνο και διότι δεν είχαμε δικαιώματα.

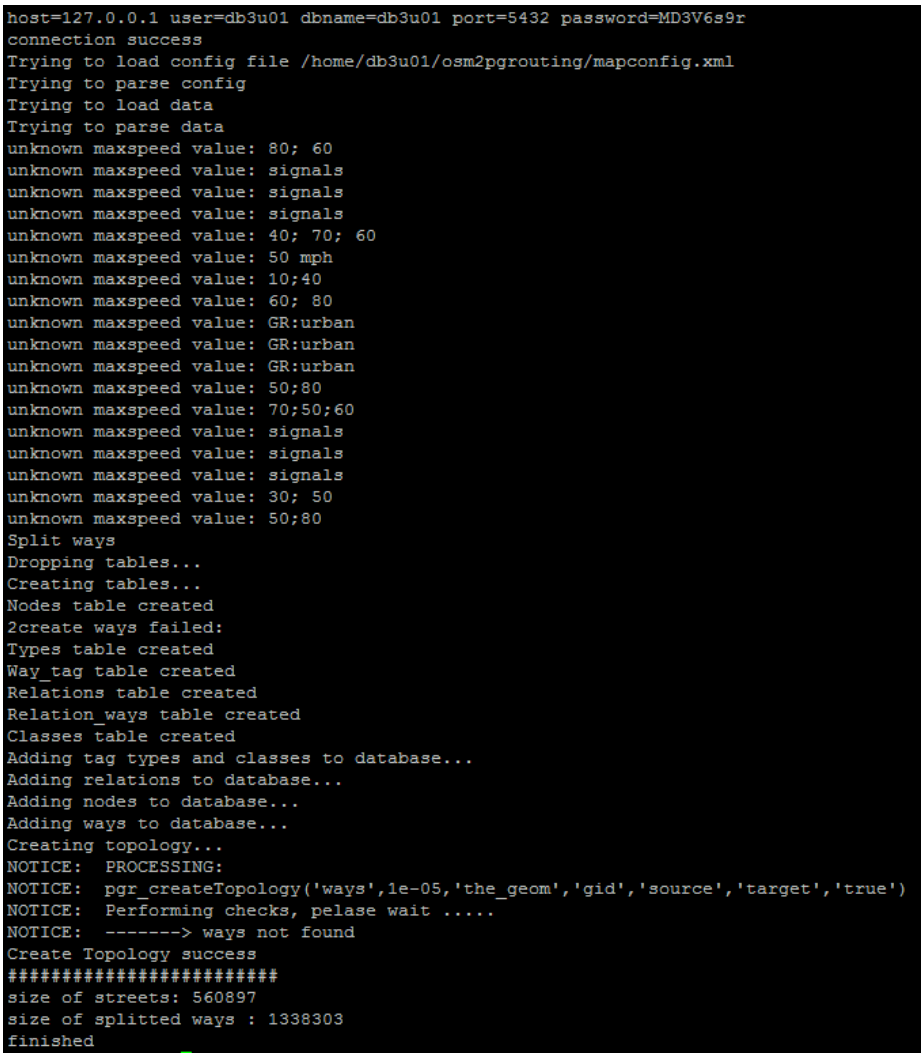
Έπειτα, χρειάστηκε να προσθέσουμε κάποια extensions στην βάση μας τα οποία χρειάζονται για να γίνει η σωστή εκτέλεση του προγράμματος osm2pgrouting τα extensions αυτά είναι τα postgis και pgRouting και προστίθενται στην βάση με τις εντολές: `psql -dbname db3u01 -c 'CREATE EXTENSION postgis'` και `psql -dbname db3u01 -c 'CREATE EXTENSION pgRouting'` αντίστοιχα τις οποίες όμως δεν καταφέραμε να τις εκτελέσουμε διότι δεν είχαμε δικαιώματα superuser, το πρόβλημα αυτό όμως λύθηκε με την δημιουργία ενός νέου template και την δημιουργία ξανά της βάσης.

Εκτέλεση του εργαλείου osm2pgrouting

Όντας στον κατάλογο μας εκτελούμε την εντολή:

```
/usr/share/bin/osm2pgrouting
-f /usr/share/osm2pgrouting/greece-latest.osm
-c /usr/share/osm2pgrouting/mapconfig.xml
-d db3u01
-U db3u01
-W MD3V6s9r
--clean
```

η οποία με τον διακόπτη -file φορτώνει το αρχείο greece-latest.osm που περιέχει το οδικό δίκτυο της Ελλάδας, με τον διακόπτη –conf φορτώνει την παραμετροποίηση, με τους διακόπτες –dbname, -user και –passwd συνδέουμε την βάση με το πρόγραμμα και ο διακόπτης –clean σβήνει το περιεχόμενο της βάσης πριν την γεμίσει. Η εκτέλεση του προγράμματος φαίνεται στην Εικόνα 4. Εδώ αξίζει να σημειωθεί πως η εκτέλεση του προγράμματος απαιτεί αρκετή μνήμη.



Εικόνα 4: Στιγμιότυπο εκτέλεσης του osm2pgrouting.

Συνοπτική περιγραφή βάσης

Όνομα πίνακα	Μέγεθος πίνακα	Πεδία πίνακα
classes	36	id, type_id, name, cost, priority, default_maxspeed
nodes	15.509.953	id, lon, lat, numofuse
osm_nodes	0	node_id, osm_id, lon, lat, numofuse, the_geom
osm_relations	0	relation_id, type_id, class_id, name
osm_way_classes	36	class_id, type_id, name, priority, default_maxspeed
osm_way_tags	1767805	class_id, way_id
osm_way_types	4	type_id, name
relation_ways	161.268	relation_id, way_id, type
relations	210	relation_id, type_id, class_id, name
relations_ways	161.268	relation_id, way_id, type
spatial_ref_sys	3.911	srid, auth_name, auth_srid, srtext, proj4text
types	4	id, name
way_tag	1.398.651	type_id, class_id, way_id
ways	1.338.303	gid, class_id, name, x1, y1, x2, y2, reverse_cost, rule,to_cost,maxspeed_forward,maxspeed_backward, osm_id, priority, the_geom, source, target
ways_vertices_pgr	1.016.615	id, osm_id,cnt, chk, ein, eout, lon, lat, the_geom, valid

Ερώτημα 3 Farthest

Στον κατάλογο `db3u01/landmark routing` υπάρχει στο αρχείο Astar.cpp υλοποίηση του αλγόριθμου Farthest σύμφωνα με τον ψευδοκώδικα του ερωτήματος 1, ακολουθει ο κώδικας σε C++:

```
int Astar::farthest(){
Timer _t;
map<node_t,Label> bfs_tree;
map<node_t, Vertex> vertexes;
queue<node_t> nodes_queue;
vector<node_t> v_gNodes;
vector<node_t> farthest_nodes;
set<node_t> result;

for(auto it=m_g->nodes_container.begin();it!=m_g->nodes_container.end();++it)  nodes_container;{
        //v_gNodes.push_back(it->n);
        bfs_tree[it->n] = Label(routing_infty , it->n);
        bfs_tree[it->n].prevNode = -1;
        bfs_tree[it->n].prevnode_idx = -1;
        vertexes[it->n] = *it;
         v_gNodes.push_back(it->n);
    }


    std::random_device rd;
```

```
    std::mt19937 generator(rd());
    vector<double> probabilities(v_gNodes.size(),1);

    // arxizo apo tixaio komvo
    std::discrete_distribution<node_t> distribution(probabilities.begin(), probabilities.end());
    auto random_node = distribution(generator);
    node_t rootNode = v_gNodes.at(random_node);


    auto lmrk_inpath_counter = 0;
    int max_level, runs, tries = 0;
    const int TRIES_LIMIT = 50;

    while (lmrk_inpath_counter < K_LMRKS && tries < TRIES_LIMIT){
        tries++;
        bfs_tree[rootNode].cost = 0;
        nodes_queue.push(rootNode);
        max_level = runs = 0;

        while (!nodes_queue.empty() ){
                runs++;
                node_t current = nodes_queue.front();
                nodes_queue.pop();

                for (auto e: vertexes[current].out_edges){
                        if (bfs_tree[e.first].cost == routing_infty){
                                bfs_tree[e.first].cost = 1 + bfs_tree[current].cost;
                                bfs_tree[e.first].prevNode = current;
                                nodes_queue.push(e.first);

                                if (max_level + 1 == bfs_tree[e.first].cost) {
                                        max_level++;
                                        farthest_nodes.clear();
                                        farthest_nodes.push_back(e.first);
                                }
                                else if (max_level == bfs_tree[e.first].cost){
                                        farthest_nodes.push_back(e.first);
                                }
                        }
                }
        }
   //cout<<"phase 2 completed. lmrk_inpath_counter: " << lmrk_inpath_counter << ", runs: " << runs
<< endl;
    rootNode = farthest_nodes.front();
    //cout << "new nodes are: ";
    if (farthest_nodes.empty()){
                //cout << "error1" << endl;
                std::discrete_distribution<node_t>             distribution(probabilities.begin(),
probabilities.end());

                    random_node = distribution(generator);
                    rootNode = v_gNodes.at(random_node);
                     //cout << "New root:" << rootNode << endl;
      }else {
                int old_size = farthest_nodes.size();
                //cout << "trying to add " << farthest_nodes.size() << " landmarks" << endl;
                        for (auto landmark: farthest_nodes){
                                result.insert(landmark);
                                if (lmrk_inpath_counter + 1 == result.size()) {
```

```
                                        //cout << "new landmark found: " << landmark << endl;
                                        lmrk_inpath_counter++;
                                }
                                if (lmrk_inpath_counter == K_LMRKS){
                                        cout << "all landmarks found!" << endl;
                                        break;
                                }
                        }
                        if (old_size == farthest_nodes.size()){
                                tries++;
                        //cout << "no landmarks new added, creating new starting point..." << endl;
    std::discrete_distribution<node_t> distribution(probabilities.begin(), probabilities.end());

                                random_node = distribution(generator);
                                rootNode = v_gNodes.at(random_node);
                                //cout << "New root:" << rootNode << endl;
                        }
                        else tries = 0;
                }
                // reset all distances
                for (auto nn: bfs_tree){
                        nn.second.cost = routing_infty;
                                if(nn.second.cost !=routing_infty){
                                        cout << "error" <<endl;
                                        break;
                                }
                }
}

for (auto it = result.begin() ; it != result.end() ; it++) m_g->landmarks_holder_avoid.push_back(*it);

if (tries == TRIES_LIMIT) std::cout << "After 50 iterations on new initial nodes the same are identical, so I give up trying..." << endl;

std::cout << "landmarks found in the path: \033[31m" << lmrk_inpath_counter << "\033[0m times" << std::endl;

std::cout << lmrk_inpath_counter << " landmarks selected in \033[31m" << _t.diff() << "\033[0ms: " << std::endl;

return 0;
}
```

Σχόλια κώδικα

Το πρόβλημα που παρατηρήσαμε κατά την υλοποίηση είναι ότι πάντα καταλήγει σε 2 φύλλα (ορόσημα - Landmarks) τα πιο απομακρυσμένα του τελευταίου επίπεδου. Αυτό σημαίνει ότι δεν υπάρχουν 16 φύλλα διαθέσιμα για να τα επιλέξει ο αλγόριθμος μας όπως είχε ζητηθεί από την εκφώνηση της άσκησης.

Γι αυτό προφανώς ίσως να ευθύνεται το το ότι τελικά ο γράφος αν και δεν είναι ισχυρά συνδεδεμένος, με τον αλγόριθμο του Tarjan's strongly connected φτιάχνετε. Θα μπορούσαμε να κινηθούμε κυκλικά έτσι ώστε να φτάσουμε στα 16 πιο απομακρυσμένα σημεία, όμως τότε ο αλγόριθμος θα ήταν ουσιαστικά λανθασμένος.

Ερώτημα 3 Planar

Σε ότι έχει να κάνει με την υλοποίηση του Planar μετά από την μελέτη σχετικής βιβλιογραφίας καταλήξαμε ότι θα πρέπει ουσιαστικά να εφαρμοστούν τα ακολουθά βήματα:

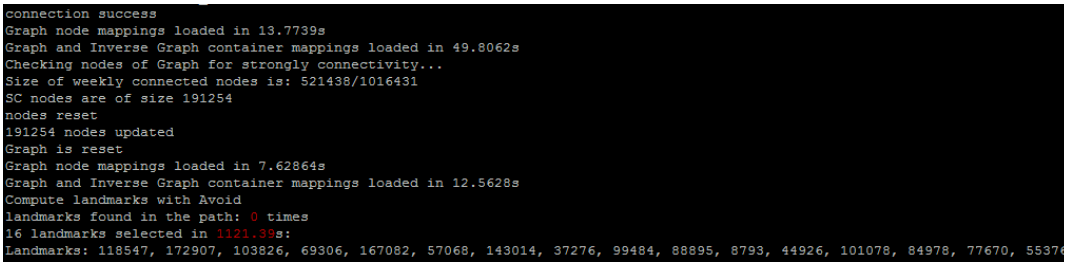
1. Μετατροπή των Καρτεσιανών συντεταγμένων (lan, lon) σε πολικές συντεταγμένες.
2. Εύρεση του κεντρικού σημείου του χάρτη.
3. Χωρίζω σε 16 τμήματα (slices) ως προς το κέντρο και για κάθε ένα από αυτά βρίσκω το σημείο με την μεγαλύτερη απόσταση από το κέντρο.

**Δεν προλάβαμε να τον υλοποιήσουμε παρότι είχαμε καταλάβει την λειτουργικότητα του.*

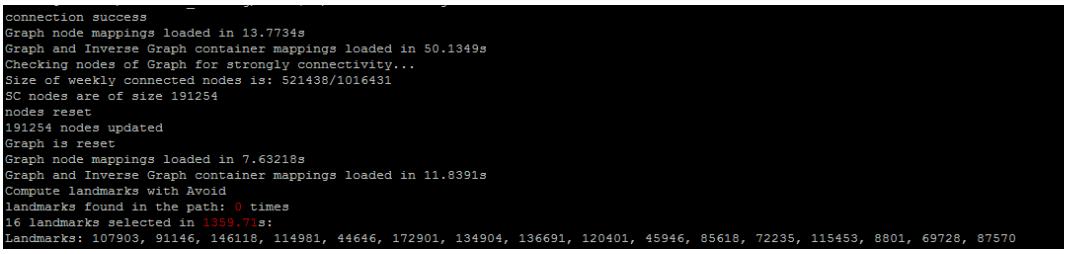
Ερώτημα 4

Χρόνοι προεπεξεργασίας

Avoid: Έπειτα από αρκετές εκτελέσεις καταλήξαμε ότι ο χρόνος εκτέλεσης για την προεπεξεργασίας των οροσήμων του Avoid κυμαίνετε από 1.120 (Εικόνα 5) έως 1.360(Εικόνα 6) δευτερόλεπτα δηλαδή περίπου 22 λεπτά της ώρας.

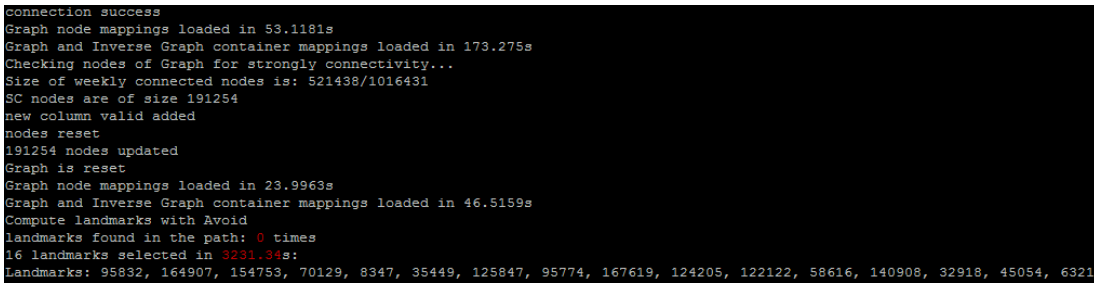


Εικόνα 5: Στιγμιότυπο εκτέλεσης Avoid.



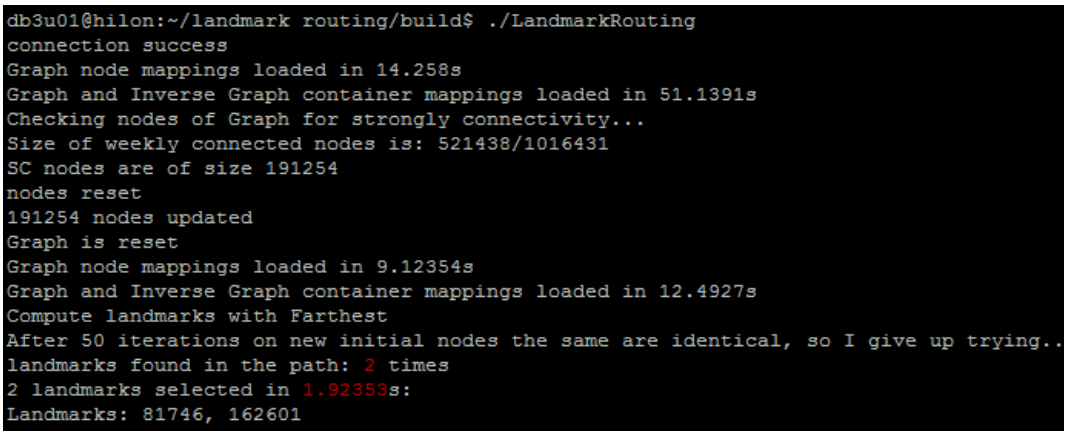
Εικόνα 6: Στιγμιότυπο εκτέλεσης Avoid.

Βεβαία παρατηρήσαμε σε μερικές περιπτώσεις χρόνους ακόμα και 3 φορές μεγαλύτερους (Εικόνα 7) από αυτούς που έχουμε αναφέρει, οι οποίοι ίσως να οφείλονται σε άλλους χρήστες οι οποίοι την ίδια στιγμή εκτελούσαν κάποια διεργασία/ες.



Εικόνα 7: Στιγμιότυπο εκτέλεσης Avoid.

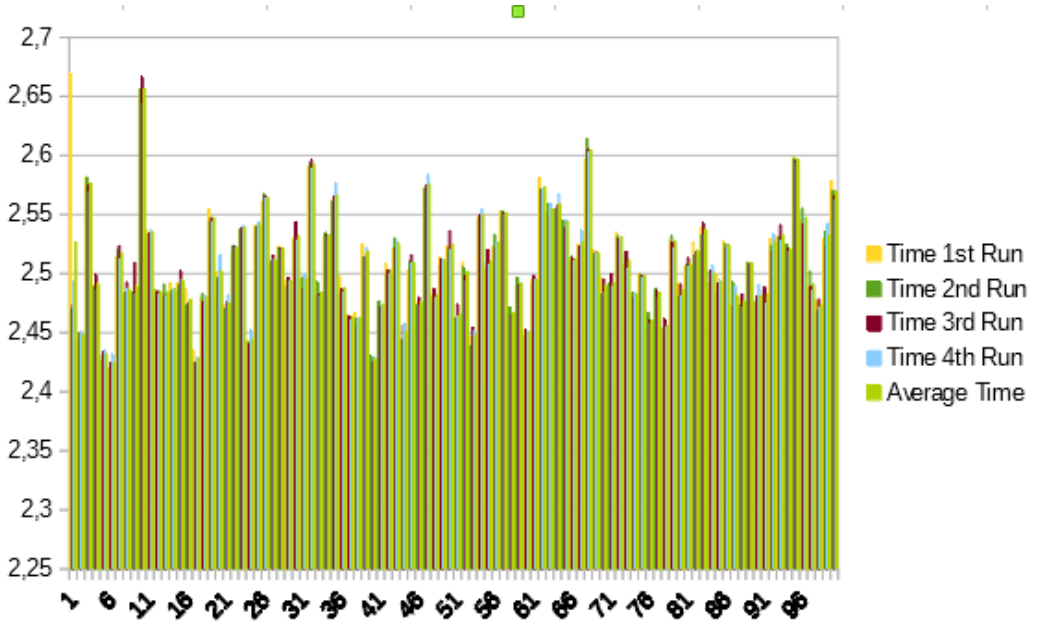
Farthest: Έπειτα από αρκετές εκτελέσεις καταλήξαμε ότι ο χρόνος εκτέλεσης για την προεπεξεργασίας των οροσήμων του Farthest είναι 1 με 2 δευτερόλεπτα (Εικόνα 8). Ο χρόνος που προκύπτει εξηγείτε στην ενότητα "σχόλια του κωδικά" του Ερωτήματος 3.



Εικόνα 8: Στιγμιότυπο εκτέλεσης Farthest.

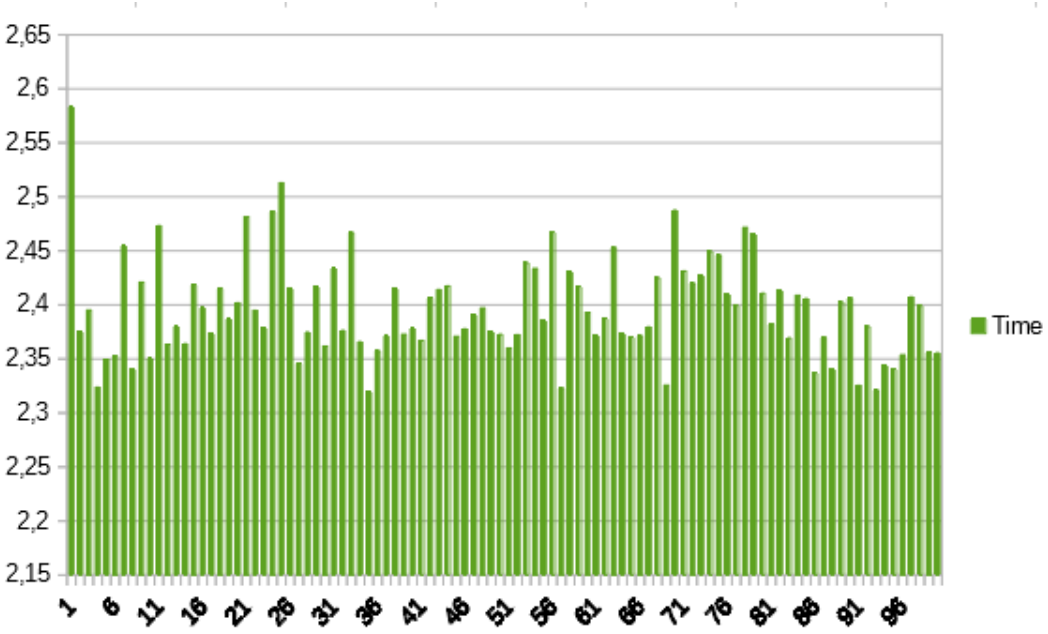
Χρόνοι εκτέλεσης του A* για 100 τυχαία ζευγάρια κόμβων

Avoid: Αναλυτικά όλοι οι χρόνοι εκτέλεσης περιέχονται στο αρχείο `Avoid_Metrics.xlsx` το οποίο περιέχει τα αποτελέσματα σε πινάκες για 3 πειράματα όπου το πρώτο (Εικόνα 9) υπολογίζει 4 φόρες τον χρόνο για κάθε ζευγάρι source-target και βρίσκει τον μέσο όρο για κάθε ζευγάρι και έπειτα αθροίζει όλους τους χρόνους (και για τα 100 ζευγάρια).

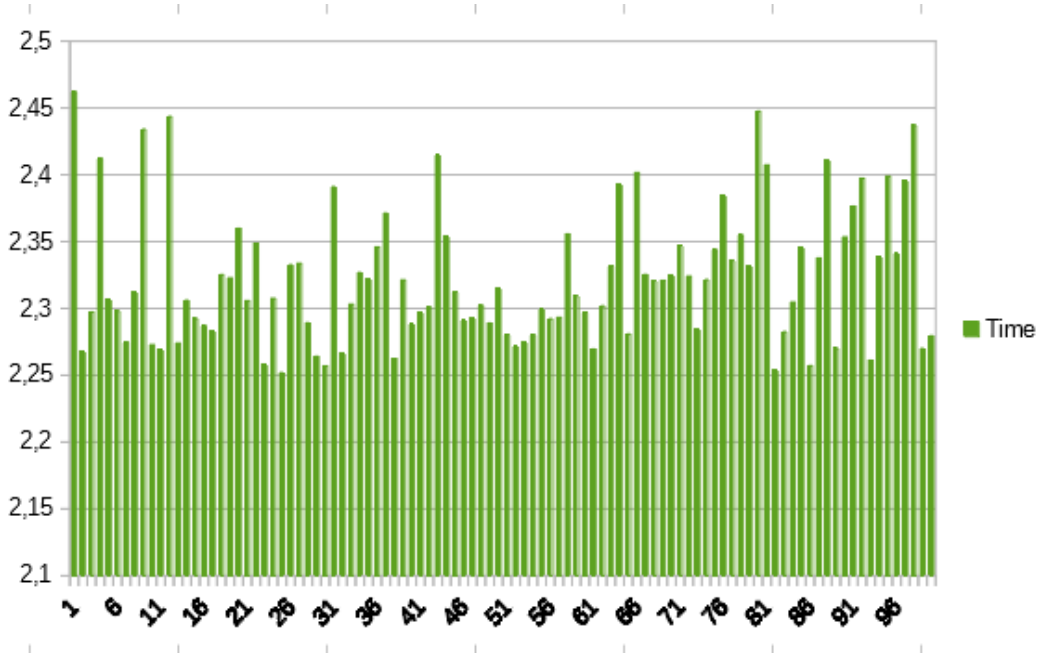


Εικόνα 9: Ραβδογράφημα 1ου πειράματος

Ενώ στα πειράματα 2 (Εικόνα 10) και 3 (Εικόνα 11) ο χρόνος κάθε ζευγαριού source-target υπολογίζεται μόνο μια φορά.



Εικόνα 10: Ραβδογράφημα 2ου πειράματος



Εικόνα 11: Ραβδογράφημα 3ου πειράματος

Από τα 3 παραδείγματα προκύπτει ότι για την εκτέλεση του A* με Avoid ορόσημα χρειάστηκαν 3,9 με 4,1 λεπτά.

Farthest: Επειδή τα ορόσημα που βρήκαμε ήταν μόνο 2 δεν επαρκούσαν ώστε να βγάλουμε αξιόπιστα αποτέλεσμα.