The EDGES spec

Oleksiy Grechnyev

February 28, 2018

Problem: Create a graph connecting all BLE beacons (vertices). Let index $i = 0, ..., N_B - 1$ number the vertices (beacons), where N_B is the number of beacons.

Step 1 : For each pair of vertices ij calculate the distance d_{ij}. If $d_{ij} >$ MAX_DIST, discard this edge. If not, include edge ij into the list (std::vector) **edges** only if there is strait white path (not crossing walls or other obstacles) between vertices i and j. The maximum distance rule is important for keeping the number of edges N_E linear in N_B for large maps. The graph edges have no direction, edge ij is equivalent to ji and included only once.

Step 2 : Now the problem is to remove the "false edges", the ones that are doubled by two or more shorter edges. First, sort the list **edges** in the *descending* order by the edge length.

Step 3 : Build the container (list of sets) **neighbors** with K_i: set of all neighbors (edge-connected vertices) of the vertex i for $i = 0, ..., N_B - 1$. The index in the sorted list **edges** serves as the edge number. **edges** and **neighbors** are two different representation of the graph: edge-based and vertex-based. Keeping both in memory is needed for linear time.

Step 4 : Loop over all edges ij starting from the longest. For each edge ij build the set of triangles named **triangles** which includes all triangles with edge ij as a side. How to do this efficiently? Find the set intersection $K = K_i \cap K_j$. All members of K are neighbors of both i and j, thus all triangles are given by $\{ijk\}, k \in K$. We have to exclude a triangle if edge ik or jk is already marked as **disabled** (see below).

Edge ij is discarded if there is at least one triangle ijk with two properties:

(a) ij is the longest side of the triangle (NOT equal !)

(b) One of the angles at vertices i or j in the triangle is smaller then CRIT_ANGLE

Variant: if there is no CRIT_ANGLE, all triangles are eliminated (except for isosceles ones), and the graph becomes tree-like with possible loops with 4 or more vertices.

The excluded edge ij is marked with a boolean flag **disabled**, and is obviously not considered for any future triangle candidates.

Step 5 : Finally, we remove all edges marked as **disabled**.

Notes:

- The algorithm presented above runs in linear time (in N_B), provided that N_E (number of edges before deletion) it linear in N_B, and the maximal number of neighbors of any vertex is bounded by a constant N_N independent on N_B. $O(N)$ time is a very good thing for an algorithm. The memory use is also $O(N)$.
- The algorithm is deterministic and independent on the vertex numbering.

Possible C++ variables:

```
struct Edge{
    int i,j; // Vertices
    double d; // Length
    bool disabled = false;
};
std::vector<Edge> edges;
std::vector<std::set<int>> neighbors;
```