July 22, 2016

Efficient Retrieval and Ranking of Maps for Geographic Queries

Andreas Behm

Efficient Retrieval and Ranking of Maps for Geographic Queries

Using the techniques of this disclosure, a system can efficiently and accurately select digital maps from a large number candidate available maps related to a specified geographic area. As discussed in more detail below, for each map, the system identifies a set of tiles (equal-sized units into which maps are divided) fully covering the map and obtains a quality metric for the map. The tiles are organized into a hierarchical data structure such as a quadtree, where each layer corresponds to a certain level of magnification. The system then generates inverted indices for the set of tiles, such that a tile is a key and the {map, quality metric} is the corresponding value. The system updates the nodes of the data structure using the indices, so that the maps subsequently can be quickly retrieved accurately and efficiently.

Availability of a large numbers of overlapping maps

Certain databases can store a very large number of maps received from individuals as well as organizations. For example, a certain user of an online digital mapping service can create and submit a neighborhood map for her home town, and a certain chain of restaurants can share a map depicting the locations of the restaurants. Because of the large number of maps, retrieving a map relevant to a certain geographic query can be computationally prohibitive. For example, if a query is submitted for maps of North America, systems utilizing conventional spatial indexing would return all maps located inside or intersecting North America. A map covering the USA and Canada would be a good match. However, a map of Mountain View, CA, although located inside of North America, would not be a good match. Accordingly, conventional systems need a huge number of maps need to be filtered in post-processing to identify relevant maps.

Representing geographic areas using map tiles

The totality of the geographic areas covered by the available maps, which can be the entire world, can be represented as a grid of tiles, each tile corresponding to a respective area of a the same fixed size. Tiles can define an hierarchy, each level of the hierarchy corresponding to a certain level of magnification. A tile at a certain level can be made up of several tiles at the next, lower zoom level. The hierarchy can be described using a directed graph or another suitable data structure. In a typical implementation, a tile at a certain zoom level corresponds to four tiles at the lower level, and the directed graph is a quadtree. For example, zoom level 0 can correspond to the root node (at which a single tile can cover all of the available geography), zoom level 1 can correspond to four child nodes of the root node (collectively covering the same area as the tile at the root node, but in more detail), etc.

The system can support of a function *parentnodes* that returns the set of parent nodes for a given node, where the set has one element in the case of a quadtree. The function returns an empty set for the root node.

An individual map represents a certain geographic area in the world. The map can be represented by a rectangle defined by pairs of coordinates specifying the lower left corner and the upper right corner, for example. Each pair of coordinate can specify latitude and longitude. More generally, maps can be represented by more complex shapes, including complex shapes and disconnected shapes.

Base quality score

The system assigns a base quality score or value to each map. The base quality score can be any suitable numerical metric that reflects such factors as the complexity of the map, whether the map comes from a trusted publisher (or, more generally, how authoritative the source of the map is), the popularity of the map, how recent the map is, etc. The base quality score is independent of a particular query.

Indexing and determining coverage

The system generates inverted indices that can be used to quickly locate suitable maps in the database. For each such inverted index, the key is a tile, and the {map identifier, quality score} tuple is the corresponding value. The values can be sorted by the map identifier or the quality score.

As part of the indexing processing, the system determines, for a certain map, a tile at the largest zoom level such that the tile is smaller than the map. The system then determines a set of tiles at this zoom level that intersect the map. This approach is schematically illustrated in Fig. 1, where the map is covered by tiles T1 and T2. The system also identifies the tile at the parent node relative to the nodes of tiles T1 and T2. As illustrated in Fig. 2, the system identifies tile T3, which covers four times the area of tile T1 or tile T2, and corresponds to the next lowest discrete zoom level.

The system then inserts a map with its base quality score into the nodes corresponding to the tiles ("tile nodes") covering the map (in this case, T1, T2, and T3). The system also multiplies each base score by a decay factor that is in the range of zero and one. Each tile defines a list of map maps with an associated score: $\{(m_1, s_1), (m_2, s_2), \ldots (m_i, s_i)\}$. The system may check whether the index already contains a map for a certain tile.

The system thus executes an indexing algorithm to insert maps into tile nodes. For a certain map, the algorithm starts with the tiles covering the map (tiles T1 and T2 in Fig. 1) and uses the base quality score. The system then identifies the one or several parent nodes for the tile (e.g., tile T3 in Fig. 2) and multiplies the base quality score by the decay factor. The system continues to traverse the quadtree upward toward the root(s) in this manner, multiplying the score at the lower level by the decay factor to gradually "decay" the quality, until at least one of the following three conditions is satisfied: (i) the tile reached is a root node, (ii) the resulting score is below a certain threshold value, or (iii) a certain threshold level within the tile hierarchy has been reached.

The following is example pseudo-code for the indexing algorithm:

```
foreach map m: M
    s = basequality(m)
    ts = region (m)
    level = 0
    while ts.size > 0 and s > threshold and level < maxlevel
        tsnext = {}
        foreach t:ts
            if not contains (t,m)
                insert(m,s) into maps(t)
            endif
            tsnext.addall(parentnodes(t))
        end foreach
        s = s*decay()
        ts = tsnext
        level++
    end while
end foreach
```

Retrieving maps

After the system populates tile nodes with maps and scores, the system can use the data structure to efficiently process queries. A query can be understood as a geographic region defined similar to a map. The system looks up maps using indices in the tile nodes that cover the query region. In order to make the result set larger, the system can retrieve the covering tiles as well, multiplying the scores by the decay factor. The results can be placed into tile-specific queues, and the resulting queues then are retrieved, fetching and removing the first element having the highest score. Fig. 3 schematically illustrates processing a query in accordance with these techniques.

The example below illustrates the processing of query *q* covering region {t1, t2} and an inverted index of five tiles {t1, t2, t3, t4, t5}:

Query *q*
region(*q*) = {t1, t2}

Tile hierarchy:
parentnodes(t1) = t3
parentnodes(t2) = t4
parentnodes(t3) = t5
parentnodes(t4) = t5
parentnodes(t5) = ()

Inverted index:
maps(t1) = <(m1, 0.9), (m2, 0.3), (m3, 0.1)>
maps(t2) = <(m1, 0.9), (m4, 0.2)>
maps(t3) = <(m5, 0.8), (m1, 0.45), (m2, 0.15)>
maps(t4) = <(m6, 0.5), (m1, 0.45), (m4, 0.1)>
maps(t5) = <(m7, 0.8), (m5, 0.4), (m6, 0.25), (m1, 0.225))
Parameters:
decay = 0.5
threshold = 0.1

The following queues are generated:

queue(t1) = <(m1, 0.9), (m2, 0.3), (m3, 0.1)>
queue(t2) = <(m1, 0.9), (m4, 0.2)>
queue(t3) = <(m5, 0.4), (m1, 0.225)>
queue(t4) = <(m6, 0.25), (m1, 0.225)>
queue(t5) = <(m7, 0.2), (m5, 0.1))

The result of processing query *q* is

retrieve(q) = <(m1, 0.9), (m5, 0.4), (m2, 0.3), (m6, 0.25), (m4, 0.2), (m7, 0.2)>

Alternatively, maps in the inverted index can be sorted by map identifies. This approach can be particularly useful for queries that involve other terms as known in a traditional search engine.

Once maps are retrieves, the system additionally can compare the retrieved maps to the query to determine an extent of the overlap, so as to tune ranking.

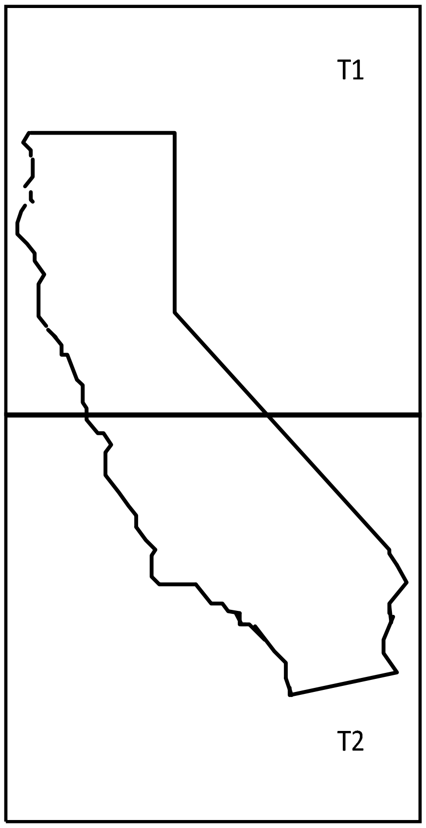

FIG. 1

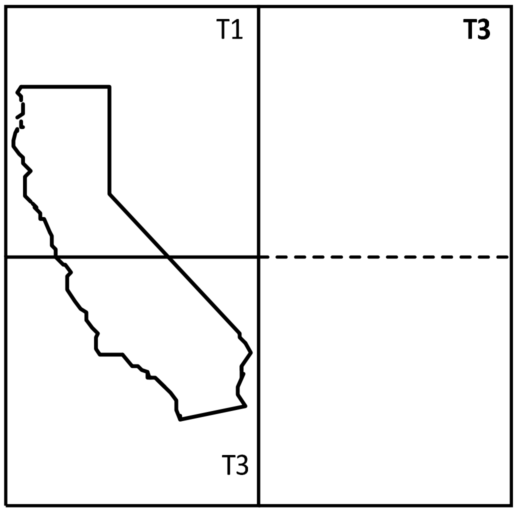

FIG. 2

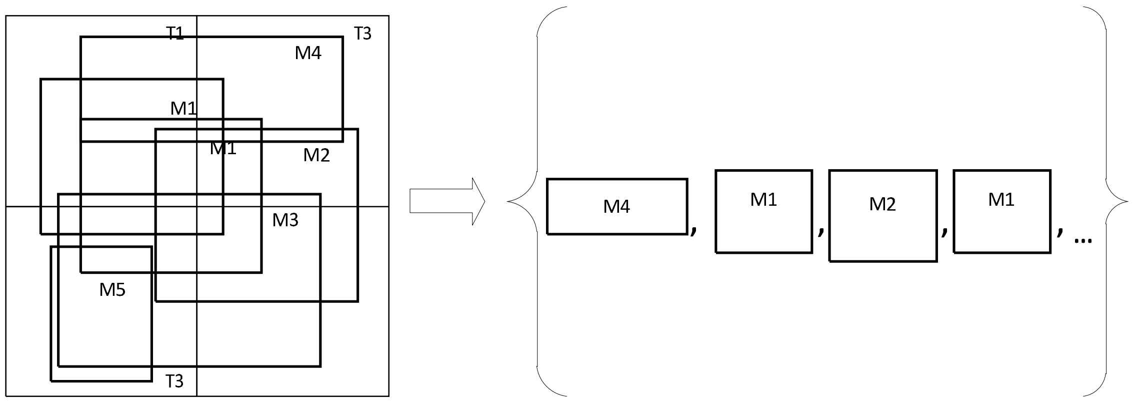

FIG. 3