Distributed Ranking Methods for Geographic Information Retrieval

Marc van Kreveld Iris Reinbacher Avi Arampatzis Roelof van Zwol

Institute for Information & Computing Sciences
Utrecht University
P.O. Box 80.089
3508 TB Utrecht
The Netherlands
{*marc,iris,avgerino,roelof*}*@cs.uu.nl*

Abstract

Geographic Information Retrieval is concerned with retrieving documents that are related to some location. This paper addresses the ranking of documents by both textual relevance and spatial relevance. To this end, we introduce *distributed ranking*, where similar documents are ranked spreaded in the list instead of sequentially. The effect of this is that documents close together in the ranked list have less redundant information. We present various ranking methods and efficient algorithms for them.

1. Introduction

The most common way to return a set of documents obtained from a Web query is by a ranked list. The search engine attempts to determine which document seems to be the most relevant to the user and will put it first in the list. In short, every document receives a *score*, or *distance to the query*, and the returned documents are sorted by this score or distance.

There are situations where the sorting by score may not be the most useful one. When a more complex query is done, composed of more than one query term or aspect, documents can also be returned with two or more scores instead of one. This is particularly useful in *geographic information retrieval* [5,6,8]. For example, the Web search could be for castles in the neighborhood of Koblenz, and the documents returned ideally have a score for the query term "castle" and a score for the closeness to Koblenz. This implies that a Web document resulting from this query can be mapped to a point in the 2-dimensional plane.

A cluster of points in this plane could be several documents about the same castle. If this castle is in the immediate vicinity of Koblenz, all of these documents would be ranked high in the sorted list, provided that they also have a high score on the term "castle". However, the user probably also wants documents about other castles that may be a bit further away, especially when these documents are more relevant for the term "castle". To incorporate this idea in the ranking, we introduce *distributed ranking* in this paper. We present various models that generate ranked lists that also have diversity. We also present efficient algorithms that compute the distributed rankings. To keep server load low, it is important to have efficient algorithms.

There are other situations where ranking based on two scores shows up: scores of two textual terms, or of a textual term and metadata information. A common example of metadata is the number of hyperlinks that link to a document. Standard in information retrieval is to combine the two scores into a single score (e.g., by a weighted sum), which produces the ranked list by sorting. Besides the problem that it is unclear how the two scores should be combined, it also makes distributed ranking impossible. Two documents with the same combined score could be similar documents or quite different. If two documents have two scores that are the same, one has more reason to suspect that the documents themselves are similar.

The topic of geographic information retrieval is studied in the SPIRIT project [5]. The idea is to build a search engine that has spatial intelligence because it will understand spatial relationships like *close to*, *to the North of*, *adjacent to*, and *inside*, for example. The core search engine will process a user query in such a way that both the term relevance and the spatial relevance of a document is obtained in a score. This is possible because the search engine will not only have a term index, but also a spatial index. These two indices provide the two scores that are needed to obtain a distributed ranking. The ranking study presented here will form part of the geographic search engine to be developed for the SPIRIT project.

Related research has been conducted in various papers [1,3,6]. They address geographic information retrieval or text document analysis, and use more than one score to rank results. The scores are combined into one score by a weighting formula. In some papers, the *additional* relevance is determined, which depends on documents ranked earlier. This is quite similar to our approach. However, we have a different starting point because we assume that we have two scores of one document based on two aspects. Furthermore, other papers do not give algorithms and efficiency analyses.

2. Distributed Ranking Methods

In this section we will present specific ranking methods. We will focus on the two dimensional case only. So we assume that a Web query has been done, and a number of relevant documents were found. Each document is associated with two scores, for example a term score and a spatial score. The relevant documents are mapped to points in the plane, and the query is also mapped to a point. We perform the mapping in such a way that the query is a point Q at the origin, and the documents are mapped to points $p_1, \ldots, p_n$ in the upper right quadrant of Q where documents with high scores are points close to Q. The point with the smallest Euclidean distance to the query is considered the most relevant document and is always first in any ranking. The remaining points are ranked with respect to already ranked points. At any moment during the ranking, we have a subset $R \subset P$ of points that have already been ranked, and a subset $U \subset P$ of points that are not ranked yet. We choose from U the "best" point to rank next, where "best" is determined by a scoring function that depends both on the distance to the query Q and on the set R of ranked points. Intuitively, an unranked point p has a higher added value or relevance if it is not close to any ranked points in R.

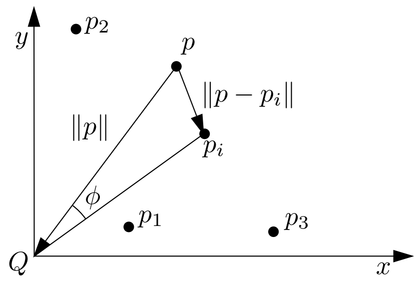

Fig. 1. An unranked point p and ranked points p_1, p_2, p_3, p_i, where p is closest to p_i by distance and by angle.

For every unranked point p, we consider only the closest point $p_i \in R$, where closeness is measured either in the Euclidean sense, or by angle with respect to the query point Q. This is illustrated by $\|p - p_i\|$ and ϕ, respectively, in Figure 1. Using the angle to evaluate the similarity of p and p_i seems less precise than using the Euclidean distance, but it allows for more efficient algorithms, and certain extensions of angle-based ranking methods give well-distributed results.

2.1. *Distance to query and angle to ranked*

Our first ranking method uses the angle measure to obtain the similarity between an unranked point and a ranked point. In Figure 1, consider the angle $\phi = \phi(p, p_i)$ and rank according to the score $S(p, R) \in [0, 1]$, which can be derived from the following normalized equation:

$$S(p, R) = \min_{p_i \in R} \left(\frac{2(\phi(p, p_i) + c)}{\pi + 2c} \cdot (\frac{1}{1 + \|p\|})^k \right) \tag{1}$$

Here, k denotes a constant; if $k < 1$, the emphasis lies on the distribution, if $k > 1$, we assign a bigger weight to the proximity to the query. The additive constant $0 < c \ll 1$ ensures that all unranked points $p \in U$ are assigned a positive score. During the ranking algorithm, we always choose the unranked point p that has the highest $S(p, R)$ score and rank it next. This implies an addition to the set R, and hence, recomputation of the scores of unranked points may be necessary. We first give a generic, quadratic time algorithm.

Algorithm 1:

(i) Rank the point p closest to the query Q first. Delete it from the point set P.

(ii) For every unranked point $p \in P$ do

  (a) Store with p the point $p_i \in R$ with the smallest angle to p.

  (b) Compute the score $S(p,R) = S(p,p_i)$.

(iii) Choose the point with the highest score $S(p,R)$ as next in the ranking; add it to R and delete it from P.

(iv) Compute for every point $p' \in P$ the angle to the last ranked point p. If it is smaller than the angle of the point stored with p', then store p with p' and update the score $S(p',R)$. Continue with step (iii).

This simple, quadratic time algorithm can easily be modified to work for different score functions.

Theorem 1 *A set of n points in the plane can be ranked according to the distance-to-query and angle-to-ranked model in $O(n^2)$ time.*

2.2. *Distance to query and distance to ranked*

We next study a model based the distance to the closest ranked point. In Figure 1, consider the distance $\|p-p_i\|$ from p to the closest ranked point p_i and rank according to the outcome of the following equation:

$$S(p,R) = \min_{p_i \in R} \left(\frac{\|p-p_i\|}{\|p\|^2} \right) \qquad (2)$$

A normalized equation such that $S(p,R) \in [0,1]$ is the following:

$$S(p,R) = \min_{p_i \in R} \left((1 - \mathrm{e}^{-\lambda \cdot \|p-p_i\|}) \cdot \frac{1}{1+\|p\|} \right) \qquad (3)$$

Here, λ is a constant that defines the slope of the exponential function. Algorithm 1 can be modified to work here as well with the same running time.

Theorem 2 *A set of n points in the plane can be ranked according to the distance-to-query and distance-to-ranked model in $O(n^2)$ time.*

2.3. *Addition models*

So far, our distributed methods were all based on a formula that divided angle or distance to the closest ranked point by the distance to the query. In this way, points closer to the query get a higher relevance. We can obtain a similar effect but a different ranking by adding up these values. It is not clear beforehand which model will be more satisfactory for users, so we analyze these models as well.

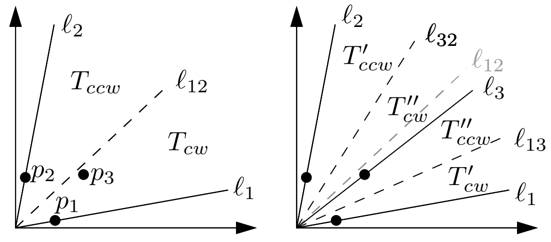

Fig. 2. The split and concatenate of trees in Algorithm 2.

$$S(p,R) = \min_{p_i \in R} \Big(\alpha \cdot (1 - \mathrm{e}^{-\lambda \cdot (\|p\|/\|p_{max}\|)}) + (1-\alpha) \cdot \phi(p,p_i) \cdot \frac{2}{\pi} \Big) \qquad (4)$$

In this equation, p_{max} is the point with maximum distance to the query, $\alpha \in [0,1]$ denotes a variable which is used to put an emphasis on either distance or angle, and λ is a constant that defines the slope of the exponential function. Algorithm 1 can be modified for this addition model, but because of the fact that the angle is now only an additive and not a multiplicative part of the score equation, we can give algorithms with better running time.

The point set is initially stored in the leaves of a binary tree T, sorted by counterclockwise (ccw) angle to the y-axis. In every leaf of the tree we also store: (i) ccw and clockwise (cw) angle to y and x-axis respectively; (ii) the distance to the query; (iii) ccw and cw score. We augment T as follows (see e.g. (Cormen et al. 1990) for augmenting data structures): In every internal node we store the best cw and ccw score per subtree. In the root we additionally store the angle of the closest ccw and cw ranked point and whether the closest ranked point is in cw or ccw direction. Additionally we store the best score per tree in a heap for quicker localization. As shown left in Figure 2, between two already ranked points p_1 and p_2, indicated by ℓ_1 and ℓ_2, there are two binary trees, cw and ccw of the bisecting barrier line ℓ_{12}. All the points in T_{ccw} are closer in angle to p_2 and all the points in T_{cw} are closer in angle to p_1. If we insert a new point p_3 to the ranking, this means we insert a new imaginary line ℓ_3 through p_3 and we need to perform the following operations on the trees:

(i) Split T_{cw} and T_{ccw} at the angle-bisectors ℓ_{32} and ℓ_{13}, creating the new trees T'_{cw} and T'_{ccw}

and two intermediate trees $\overline{T}_{cw}$ and $\overline{T}_{ccw}$

(ii) Concatenate the intermediate trees from (i), creating one tree $\overline{T}$.

(iii) Split $\overline{T}$ at the newly ranked point p_3, creating T''_{cw} and T''_{ccw}.

Figure 2 right, shows the outcome of these operations. Whenever we split or concatenate the binary trees we need to make sure that the augmentation remains correct, which is standard. We also need to update the information in the root of each tree about the closest cw and ccw ranked point and the best scores. As the scores are additive, and all scores for points in the same tree are calculated with respect to the same ranked point, we simply subtract the cw (ccw) angle of the closest ranked point from the cw (ccw) best score to get the new best score for the tree. Furthermore we need to update the score information in the heap. Now we can formulate an algorithm for the addition-model that runs in optimal $O(n \log n)$ time.

Algorithm 2:

(i) Create T with all points of P, the augmentation and a heap that contains only the point p closest to the query Q.

(ii) Choose the point p with the highest score $S(p, R)$ as next in the ranking by deleting the best one from the heap.

(iii) For every last ranked point p do:

(a) Split and concatenate the binary trees as described above and update the information in their roots.

(b) Update the best-score information in the heap: delete the remaining best score of the old tree that did not contain p and insert the four best scores of the new trees.

Theorem 3 *A set of n points in the plane can be ranked according to the angle-distance addition model in $O(n \log n)$ time.*

Another, similar, addition model adds up the distance to the query and the distance to the closest ranked point. Algorithm 2 is not applicable for this addition model, because the distance to the closest ranked point does not change by the same amount for a group of points as the angle. This implies that the score for every unranked point needs to be adjusted individually when adding a point to R. We can modify Algorithm 1 for this addition model. Alternatively, we can use an algorithm that has $O(n^2)$ running time in the worst case, but a typical running time of $O(n \log n)$, as in [4,7]. The idea is to maintain the Voronoi diagram of the ranked points, and with each Voronoi cell, maintain the unranked points in it. This practically fast algorithm can be applied to all ranking methods described thus far.

3. Conclusions

This paper introduced distributed relevance ranking for documents that have two scores. In the full paper we present more models and also several extensions of the models. We have implemented the rankings to inspect how well they spread, while not sacrificing distance to the query too much. It seems that certain extensions of the models presented here perform best. However, user evaluation is needed (and is planned) to test this properly.

References

[1] J.G. Carbonell and J. Goldstein. The use of MMR, diversity-based reranking for reordering documents and producing summaries. In *Research and Development in Information Retrieval*, pages 335–336, 1998.

[2] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Press, Cambridge, MA, 1990.

[3] J. Goldstein, V.O. Mittal, J.G. Carbonell, and J.P. Callan. Creating and evaluating multi-document sentence extract summaries. In *Proc. CIKM*, pages 165–172, 2000.

[4] P. S. Heckbert and M. Garland. Fast polygonal approximation of terrains and height fields. Report CMU-CS-95-181, Carnegie Mellon University, 1995.

[5] C.B. Jones, R. Purves, A. Ruas, M. Sanderson, M. Sester, M.J. van Kreveld, and R. Weibel. Spatial information retrieval and geographical ontologies – an overview of the SPIRIT project. In *Proc. 25th Annu. Int. Conf. on Research and Development in Information Retrieval (SIGIR 2002)*, pages 387–388, 2002.

[6] E. Rauch, M. Bukatin, and K. Naker. A confidence-based framework for disambiguating geographic terms. In *Proc. Workshop on the Analysis of Geographic References*, 2003.

[7] M. van Kreveld, R. van Oostrum, and J. Snoeyink. Efficient settlement selection for interactive display. In *Proc. Auto-Carto 13: ACSM/ASPRS Annual Convention Technical Papers*, pages 287–296, 1997.

[8] U. Visser, T. Vögele, and C. Schlieder. Spatio-terminological information retrieval using the BUSTER system. In *Proc. of the EnviroInfo*, pages 93–100, 2002.