



Spatially-Aware Information Retrieval on the Internet

SPIRIT is funded by EU IST Programme
Contract Number: IST-2001-35047



Abstract Multi-Attribute Similarity Ranking

Deliverable number:	D17:5301
Deliverable type:	R
Contributing WP:	WP 5
Contractual date of delivery:	1 April 2004
Actual date of delivery:	13 April 2004
Authors:	Avi Arampatzis, Marc van Kreveld, Iris Reinbacher
Keywords:	Similarity Ranking, Relevance Ranking

Abstract: In the previous two deliverables we introduced, firstly, simple geometric formulas to calculate spatial scores between footprints, and secondly, ways to combine spatial with textual scores in order to generate relevance rankings. The methods were implemented and integrated in the interim prototype of the SPIRIT search engine. In this report we present ideas on how to improve on both spatial scores and score combination methods.

Contents

1.	INTRODUCTION	3
2.	SPATIAL SCORES REVISITED	4
2.1.	Alternative measures of distance.....	4
2.2.	Multiple Document Footprints	4
3.	EXTENSIONS ON COMBINING SCORES.....	5
3.1.	Staircase Enforced	6
3.2.	Limited Windows	7
3.3.	Higher Dimensions	8
	REFERENCES.....	9

D17:5201

Abstract Multi-Attribute Similarity Ranking

1. Introduction

For high-quality ranking two things are required. Firstly, we need a good spatial score between query and document footprints. Secondly, we need a good combination of the spatial and textual (BM25) scores.

Obtaining a spatial score has been discussed in a preliminary sense in a previous deliverable [1]. Here, the spatial relationships (distance, containment, and direction) were converted into numeric values that indicate how close, how much inside, or how much North-of the relationship between two objects is. Those numeric values were first attempts at obtaining a score to quantify spatial relationships. However, they did not incorporate context. For example, let us assume three cities, A, B, and C, where A lies in equal distance (in a Euclidean sense) from B and C. If C is bigger than B, then the score of B being close to A should be lower than that of C being close to C. In other words, the distance scores of cities around A may depend on the context, i.e. which other cities are around A. Also, natural barriers can influence the concept of proximity. It matters a lot whether a distance of 10 km (as the crow flies) can be covered by a direct road, or requires a large detour around a mountain range (or a small road over a mountain pass). A few ideas on how these contextual problems are presented in Section 2.1.

Another weakness of our methods lies in the way we treat multiple-footprint documents. While we assume that a query can have only one footprint (a user is interested in only one location), documents may have multiple footprints (refer to more than one location). The method we followed so far in order to calculate the spatial score considers only the best-matching document footprint. For example, if a user is looking for “airports near London”, a document that refers to both “Gatwick” and “Stansted” is scored as referring only to “Gatwick” since it’s the nearest airport of the two. Such a document, however, should be scored higher than another that refers only to “Gatwick” since it provides more relevant information. Moreover, initial tests of the interim prototype pointed to the need of some form of *document length normalization*, in the sense of the number of footprints occurring: Gatwick’s official web-pages should be more important than a web-list of all airports in UK.

Obtaining a good combination of different scores, such as a spatial score and a textual score was already discussed to some extent in a previous deliverable [2]. Here we extend upon the ideas presented there, and introduce the staircase enforced variation on those methods. Also, we introduce the limited windows extension of those methods. Both extensions can be expected to provide a better combination of scores, and hence, a better ranking.

This document elaborates on these two aspects of obtaining better rankings. They are discussed in the next two sections of this document.

2. Spatial Scores Revisited

2.1. Alternative measures of distance

In the previous two deliverables ([1] and [2]) we considered distance in the Euclidean sense. There are cases, however, where it may not be the best choice. From a human perspective, alternative measures of distance¹ may have to consider:

- more important objects of the same type closer by than the specified one (cities example in the previous section),
- difficulty of transportation influences distance,
- crossing borders may influence distance,
- distance to a region can be distance to the closest point of that region, but it can also be “average” distance to some random point in that region.

Alternative measures of directional relationships:

- extensions of the ones with the distance concepts above.

Alternative measures of containment:

- containment seems a true/false condition, but can be put into a value,
- a non-point object (river, road, forest) can be partially inside a region-object,
- a point or non-point object can be really inside (close to the middle) or inside and close to the boundary,
- in case of objects with indeterminate boundaries, containment should be treated differently too. We can work with certainly inside, probably inside, most likely not inside, etc. (qualitative relationships).
- Containment as obtained from a containment hierarchy can also use the number of intermittent levels: Amsterdam is in Europe, and the Netherlands is in Europe, but the Netherlands is higher up in the hierarchy below Europe.

We are currently investigating whether the above extensions are possible, given the information provided by the ontology. In following deliverables we will be able to define some of them in a more concrete way.

2.2. Multiple Document Footprints

In this section we are going to incorporate two pieces of information into the way that a spatial document score is calculated:

- The number n of unique footprints in a document.
- The frequencies f_1, \dots, f_n , of occurrence of the footprints in the document.

Moreover, the total spatial score of a document will be derived from fractional score contributions of all occurring document footprints.

¹ The term “distance” can be reserved for geometric distance, whereas the term “closeness” can be used for distance as used by users and is a more qualitative concept.

A simple way of taking into account all document footprints is to define the total spatial score as a linear combination (e.g. the simple average) of the individual scores of the footprints:

$$S = 1/n * (s_1 + \dots + s_n),$$

where s_i is the score of the i th document footprint in respect to the query footprint and is calculated as defined in [1]. Incorporating also the frequencies of occurrence f_i , let us define the weight of a footprint as:

$$tf_i = 1 + \log(f_i).$$

A footprint that occurs in the document only once will get a weight of one, where any extra occurrences will increase the weight in a log fashion. The total score may be calculated as

$$S = 1/(tf_1 + \dots + tf_n) * (tf_1 * s_1 + \dots + tf_n * s_n),$$

that is the weighted average of the individual scores. Considering again the example in the Section 1 about “airports near London”, such a scoring function like the last one would score higher Gatwick’s official web-page than a web-list of all UK airports. Moreover, it takes into account more than the best-matched document footprint.

The last formula may serve as a starting point for improving the spatial scoring function. Empirical data are necessary for revealing weaknesses and directions for further improvements and fine-tuning.

3. Extensions on Combining Scores

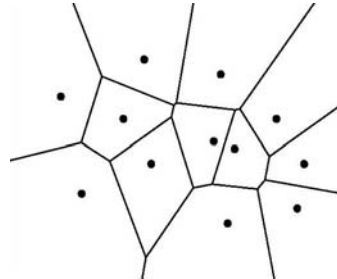
In this section, we present extensions of the basic ranking methods introduced in [2]. We divided the given point set P into the set of already ranked points R and the set of unranked points U during the algorithms’ running. We chose the candidates to be ranked next from all points in U and computed the score functions with respect to all ranked points in R . In this section we present two extensions where this is not the case anymore. The staircase enforcement (Section 3.1) limits the set of candidates to be ranked next to the points in U that lie on the staircase of the point set P . The limited windows method limits the set of ranked reference points to be used for the score computation to only the last k ranked points. Furthermore, we will describe the necessary adaptations of the algorithms in higher dimensions.

First we introduce a variation of Algorithm 1 presented in [2]. It uses a Voronoi diagram (see following figure) to find the closest unranked points to a ranked point p . Its worst case running time is the same as of Algorithm 1, namely $O(n^2)$; however, a typical case analysis shows that it generally runs in $O(n \log n)$ time in practice.

Algorithm 2: Given a set P with n points in the plane.

1. Rank the point p closest to the query Q first. Add it to R and delete it from P . Initialize a list with all unranked points.
2. For every newly ranked point p in R do:
 - a) Insert it to the Voronoi diagram of R .

- b) Create for the newly created Voronoi cell a list of unranked points that lie in it by taking those points that have p as the closest ranked point from the lists of the neighboring cells. For all R' (subsets of R) Voronoi cells that changed, update their lists of unranked points.
 - c) Compute the point with the best score for the newly created Voronoi cell and insert it in a heap H . For all R' (subsets of R) Voronoi cells that changed, recompute the best score and update the heap H accordingly.
3. Choose the point with the best overall score from the heap H as next in the ranking; add it to R and delete it from P and H .
 4. Continue with step 2.



Example Voronoi diagram

Since the average degree of a Voronoi cell is six in two dimensions, one can expect that a typical addition of a point p to the ranked points involves a set R' with not many more than six ranked points. If we also assume that, typically, a point in R' loses a constant fraction of the unranked points in its list, we can prove an $O(n \log n)$ time bound for the whole ranking algorithm.

3.1. Staircase Enforced

In the basic methods, every unranked point was eligible to be next in the ranking, if it had the highest score. This can lead to a ranking where points (documents) which are better in both aspects (spatially and textually) can be ranked after a point which is further away from the query. This is a rather undesirable outcome. As an alternative, we choose the candidates to be ranked next only from the points p that lie on the staircase of the point set. A point p is on the lower left staircase of the point set P if and only if for all p' in P , we have $p_x < p'_x$ or $p_y < p'_y$. Such a ranking automatically has the property that any ranked document is more relevant in at least one aspect than all documents that were ranked later.

We can easily adapt the basic ranking algorithms given above for the staircase enforcement. There are two main ways to do that: First, we can restrict the input set for the algorithm to only those points p that lie on the staircase or, second, we can use the whole point set P as before, but now we consider as eligible candidates for next in the ranking only the points on the staircase. In both cases, after a point gets ranked, the staircase has to be updated and points that are new on it either have to be inserted in the input set or marked as eligible. Computing the staircase of a point set with n point takes $O(n \log n)$ time, insertion or deletion of one point takes $O(\log n)$ time per point. In every update of the staircase, we delete one point from it. There can be as many as a linear number of points that need to be inserted in one update, so a single update can have a running time of $O(n \log n)$. However, every point of P is inserted to the staircase and deleted from it exactly once. That means that maintaining the staircase takes only $O(n \log n)$ time during the whole algorithm. A brief description of the necessary changes to the basic algorithms follows.

In the generic Algorithm 1 (see [2]), which can be used for all models, when using the second adaptation, we compute the staircase at the beginning, and then we only need to change Step 3 as follows:

- Update the staircase of P and choose from the points on the staircase the one with highest score $S(p,R)$ as next in the ranking; add it to R and delete it from P and from the staircase.

Computing the staircase in the beginning takes $O(n \log n)$ time. We only need to do this once. Updating the staircase in Step 3 takes $O(\log n)$ time per insertion and deletion, and the other computations can be performed in constant time. As this step needs to be repeated n times, it takes $O(n \log n)$ time overall, and hence, staircase enforcement does not influence the total running time of $O(n^2)$.

In the adaptation of Algorithm 2 for the angle and distance model, besides the computing of the staircase at the beginning of Step 2, the only change is that the points with maximum score $S(.,.)$ that are inserted in the heap, must lie on the staircase. That means that all points on the heap (one point per Voronoi cell) lie on the staircase of P . Neither the worst case nor the typical time analysis change in this case, so the worst case time bound is still $O(n^2)$, and the typical running time remains $O(n \log n)$.

3.2. Limited Windows

In all the previously presented basic ranking methods as well as in the staircase enforced extensions, the relative closest point r from the whole set R of already ranked points was used to determine the score of an unranked point p . However, if the second best point lies really close to the first, it should still be chosen fairly early. Therefore, we present the limited windows extension, where only the k latest ranked points are kept in evidence for the future ranking, where k is fixed throughout the algorithm.

The general idea for this method is as follows. We store the set of ranked points that we consider in a queue W , which we will call the window. The basic algorithms remain exactly the same as long as the size of W does not exceed k . When the $(k+1)$ point is ranked, we add it at the tail of the queue and delete the point p at the head of it. For all unranked points that had the deleted point p as their closest, we need to find the new closest point among the k ranked points in W and to re-compute their score.

We give a brief description of the necessary adaptations of the basic algorithms in the next paragraphs. An adaptation of the staircase enforced methods to get a combination of the two extensions is straightforward and therefore not given here. We assume that k is fixed.

The adaptation of Algorithm 1 is straightforward. We now determine in Step 4 (Algorithm 1) for all unranked points p' the smallest distance to the k last ranked points in R and compare it to the distance that is stored with p' . If it is smaller than the distance to the point stored with p' , the new closest point p is stored with p' , and the score $S(p',R)$ is updated as before. As we enlarge the set of ranked points, we compute the distance to the last k , the new step 4 takes $O(kn)$ time, so the overall running time of the algorithm is now $O(kn^2)$. In fact, the very simple brute-force method also takes $O(kn^2)$ time here. If we allow $O(kn)$ extra storage, we can store with every unranked point all k last ranked points sorted by distance. We can keep the sequence of all k points sorted in $O(n^2 \log k)$ time for all points.

In Algorithm 2 for the angle or distance model, additionally to the basic algorithm, we add every ranked point to a queue W . As soon as W contains more than k points, we delete the first point p from the queue and from the Voronoi diagram of R . All unranked points that lay in the Voronoi cell of p need to be redistributed to the at most k neighboring cells, which means that their lists of unranked points need to be updated. Furthermore, we need to re-compute the highest score value from these k lists and update the heap H accordingly.

The worst case time analysis for deleting one point from the window is as follows. The operations on W only take constant time. Deleting the point p from the Voronoi diagram can be done in $O(k)$ time. There can be a linear number of unranked points that need to be redistributed, which takes $O(kn)$ time in the worst case. Updating the k lists can be done in linear time and updating the heap takes $O(k \log k)$ time. This leads to an overall worst case running time of $O(kn^2)$.

In a typical time analysis we can delete the point from the queue and the Voronoi diagram in constant time. Redistributing the unranked points to the neighboring cells and updating the lists accordingly takes $O(n/k)$ time. Updating the heap takes $O(\log k)$ time typically. Overall we therefore get a typical running time of $O((n^2)/k + n \log k)$.

3.3. Higher Dimensions

So far, we have considered only two possible scores that are combined to give a distributed ranking. Naturally, we can have more than two scores for the point set P , and the two presented methods can be used for any number of scores. This is equivalent to extending the basic algorithms to higher dimensions. In this section we will briefly discuss the extensions of the presented algorithms to higher dimensions.

The generic Algorithm 1, which can be applied to both presented models (Sections 3.1 and 3.2), works as is in any dimension d . Therefore, we conclude that the worst case running time for the generic algorithm is $O(n^2)$ in any dimension. In fact, this algorithm is optimal for higher dimensions $d \geq 3$.

Algorithm 2 for the basic models can easily be extended to higher dimensions $d \geq 3$. We still need to maintain the Voronoi diagram for the ranked points and the lists of unranked points that lie in each Voronoi cell. The point with the highest score value per cell is stored in a heap, where the next point to be ranked is chosen from.

In three dimensions, the Voronoi diagram for the distance method is the normal Voronoi diagram. It can be computed in $O(n^2)$ worst case time. The Voronoi diagram for the angle model in three dimensions can be determined as follows: we project the ranked points to the surface of the unit sphere and compute the Voronoi diagram on this surface, which leads to a worst case running time of $O(n^2)$. In higher dimensions d the worst case running time is $O(n^{\lceil d/2 \rceil})$. Analyzing the typical running time for this algorithm in higher dimensions is more difficult.

Our ranking algorithm chooses the points such that they are well spread, so we can assume that the ranked points are inserted to the Voronoi diagram in a randomized order. If we could furthermore assume that the point set we are dealing with, is nicely distributed as well, the typical running time for any dimension d would be $O(n \log n)$. However, if we are looking at a point set that is derived from a web query, it is very likely that it consists of several dense clusters. That means, that we cannot guarantee that only a constant number of Voronoi cells is changed when a new point is ranked, especially in the later stages of the algorithm. This is the main obstacle on our way to the typical running time, so all we can prove is the worst case running time of $O(n^{\lceil d/2 \rceil})$.

References

[1] *Simple and Useful Similarity Measures*, Avi Arampatzis and Marc van Kreveld, SPIRIT Report D9:5102, September 2003.

[2] *Simple Similarity Ranking Procedure*, Avi Arampatzis, Marc van Kreveld, and Iris Reinbacher, SPIRIT Report D14:5201, January 2004.