Proximity Search in Databases

Roy Goldman, Narayanan Shivakumar,
Suresh Venkatasubramanian, Hector Garcia-Molina

Stanford University
{royg, shiva, suresh, hector}@cs.stanford.edu

Abstract

An information retrieval (IR) engine can rank documents based on textual proximity of keywords within each document. In this paper we apply this notion to search across an entire database for objects that are "near" other relevant objects. Proximity search enables simple "focusing" queries based on general relationships among objects, helpful for interactive query sessions. We view the database as a graph, with data in vertices (objects) and relationships indicated by edges. Proximity is defined based on shortest paths between objects. We have implemented a prototype search engine that uses this model to enable keyword searches over databases, and we have found it very effective for quickly finding relevant information. Computing the distance between objects in a graph stored on disk can be very expensive. Hence, we show how to build compact indexes that allow us to quickly find the distance between objects at search time. Experiments show that our algorithms are efficient and scale well.

1 Introduction

Proximity search is successfully used in information retrieval (IR) systems to locate documents that have words occurring "near" each other [Sal89]. In this paper we apply this notion to search across an arbitrary database for objects that are "near" other objects of interest. Just as the distance between words in a document is an approximation of how related the terms are in the text, proximity search across an entire database gives a rough or "fuzzy" measure of how related objects are. While some situations demand "precise" query results, more and more online databases—such as content databases on the Web—enable users to interactively browse results and submit refining queries. In these settings, proximity estimates can be very useful for *focusing* a search. For example, we may be looking for a "person" with a last name that sounds like "Schwartz" but may not know if this person is an employee, a manager, or a customer. A search may yield many people, spread out throughout the database. If we also know that the target person is somehow related, say, to a particular complaint record, then we can narrow down the original set, ranking it by how closely related each person is to the complaint. Similarly, in a database that tracks overnight package delivery, we may wish to locate any information pertinent to a lost package (e.g., people that handled it, locations it went through, customers that signed for it) ranked by how relevant the information is to the lost package.

**Proceedings of the 24th VLDB Conference
New York, USA, 1998**

For object-proximity searching, we view the database simply as a collection of objects that are related by a *distance* function. The objects may be tuples, records, or actual objects, or even fields within these structures, if finer granularity is desired. The distance function is provided by the system or an administrator; it indicates how "closely related" certain (not necessarily all) pairs of objects are. For instance, in a personnel database, the number of links that separate objects may be a good measure of how closely they are related. Two employees working in the same department are closely related (each employee is linked to the same department); if two departments cooperate on the same product, then an employee in one department is related to an employee in the other, but to a lesser extent. We can also weight each type of link to reflect its semantic importance. In a relational context, tuples related by primary-key/foreign-key dependencies could be considered closely linked, while tuples in the same relation could also be related, to a lesser extent.

Traditional IR proximity search is intra-object, i.e., it only considers word distances within a document. Our search is inter-object, i.e., we rank objects based on their distance to other objects. This difference introduces two related challenges, which are the main focus of this paper.

- *Distance Computation*: Text intra-object distance is measured on a single dimension. Thus, it is easy to compute distances between words if we simply record the position of each word along this one dimension. For inter-object search, we measure distance as the length of the shortest path between objects.

- *Scale of Problem*: For efficient inter-object proximity search, we need to build an index that gives us the distance between *any* pair of database objects. Since there can be a huge number of objects, computing this index can be very time consuming. For intra-object search, on the other hand, we only need to know the distance between words within an object, a much smaller problem.

In this paper we describe optimizations and compression schemes that allow us to build indexes that can efficiently report distances between any pair of objects. Experiments show that our algorithms have modest time and space requirements and scale well.

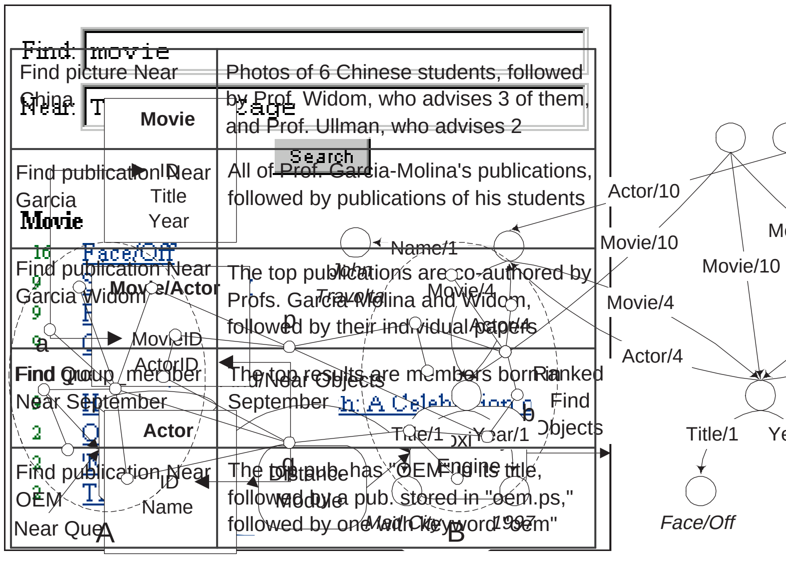
In Section 2, we trace an example over a sample database, to further motivate inter-object proximity search. Section 3 then defines our problem and framework in more detail. In Section 4, we illustrate a particular instance of our general framework, as applied to keyword searching over databases. Section 5 details our algorithms for efficient computation of distances between objects, and experimental results are given in Section 6. We discuss related work in Section 7.

2 Motivating Example

The Internet Movie Database (www.imdb.com) is a popular Web site with information about over 140,000 movies and over 500,000 film industry workers. We can view the database as a set of linked objects, where the objects represent movies, actors, directors, and so on. In this application it is very natural to define a distance function based on the links separating objects. For example, since John Travolta stars in the movie "Primary Colors," there is a close relationship between the actor and the movie; if he had directed the movie, the bond might be tighter.

Within our framework, proximity searches are specified by a pair of queries:

- A *Find query* specifies a *Find set* of objects that are potentially of interest. For our example, let us say that the find query is keyword-based. For instance, "*Find* movie" locates all objects of type "movie" or objects with the word "movie" in their body.



Find picture Near China	Photos of 6 Chinese students, followed by Prof. Widom, who advises 3 of them, and Prof. Ullman, who advises 2
Find publication Near Garcia	All of Prof. Garcia-Molina's publications, followed by publications of his students
Find publication Near Garcia Widom	The top publications are co-authored by Profs. Garcia-Molina and Widom, followed by their individual papers
Find group_member Near September	The top results are members born in September
Find publication Near OEM	The top pub. has "OEM" in its title, followed by a pub. stored in "oem.ps," followed by one with keyword "oem"