Expert Finding Systems

R&D Project Report

Bachelor of Technology (Honours)

by

Pratyaksh Sharma
Roll No : 120050019

Collaborators: Manoj K. Agarwal
Prof. Krithi Ramamritham

Department of Computer Science and Engineering
Indian Institute of Technology Bombay
Mumbai 400076, India

November, 2015

Abstract

Expert finding is an important problem studied in the context of domains ranging from academic to legal. With the ultimate goal of developing a web-scale service to answer queries of the type "Who is the best expert in X", we take a clean-slate approach and study a number of challenges not satisfactorily addressed by current approaches. We propose a preliminary model that provides the ability to seamlessly absorb dynamically available heterogeneous information. In addition, we study the problem of *coverage* in the query result and propose appraches to solve it.

Contents

Chapter 1

Introduction

Expert finding is a frequently occurring problem in a number of applications and has been studied widely. Stated tersely, given a topic T, it is the task of finding a ranked list of people who are knowledgeable in T. The precise definition of 'knowledgeable' differs with context and the solutions usually strive for approaches that can work with a any of them.

The problem occurs in a number of domains, for example, the medical domain in which the problem is to find experts in treating a particular medical condition or experts in a sub-speciality of medicine. Analogously, in the legal domain, the expert-finding information need would be to find legal practitioners who are experts in a given sub-field of law.

Despite the fragmentation of the problem by various domains, the work in the area has been largely consolidated towards tackling the problem in the academic domain. Since the academic process is highly social, the academic community is a particularly well suited model for the study of expert-finding systems. With relationships such as co-authorship and supervisor-of between academics, a measure of expertise needs to take into account the interaction of the academic with her peers. We adopt the academic framework for studying the problem, though we expect that the ideas will be fairly general and would adapt naturally to other domains.

Even when restricted to a network of academics, there are several problems that have not been sufficiently addressed by prior work in the area. First, the information that will be used by a system can come from various different sources. Further, it may so happen that differing sets of information is available for each academic. For example, for one academic, the information about her publications might completely specified but her affiliation with an institute may not be. In such a case, the system should be able to work with whatever information is available for each academic.

Given an expert-finding query, it could so happen that the top results may be confined to certain closely-related persons. For example, all the top experts in "databases" could come from Cucumber and Mellon University, but the user may desire that the results have better representation from other universities as well. We call this the problem of coverage,

and it is one of the central themes in our approach.

Owing to the large size of academic networks and the availability of information updates at an increasingly fast pace, an expert-finding system should ideally be able to incorporate the new information in real time. Existing approaches that assume a static network for analysis would require large re-computation on even small changes in the network structure. We strive towards a model that would need to perform only small local-computations to absorb updates in the network structure.

In the light of the above goals, we discuss an approach towards building an expert-finding system that focuses on the following aspects:

1. seamless integration with heterogeneous data sources available,
2. providing coverage guarantees in the query result, and
3. efficiently answering queries even in case of highly dynamic graphs.

1.1 Outline

In chapter 2 we give an overview of the complete system, to give the reader a better sense of the overall requirements and functionality. Chapter 3 provides a somewhat detailed discussion on query processing, in which we propose our scoring model that will ultimately lead to expert-ranking. Finally, chapter 4 summarises the goals achieved so far and discusses future work.

Chapter 2

Architecture

In this chapter we discuss the overall anatomy of an expert-finding system that shall form the background of our exploration in the rest of the chapters.

2.1 Layered Architecture

As we shall see shortly, our system should be capable of performing several tasks and that these tasks are fairly independent. It is natural to isolate independent tasks into separate layers for the purpose of studying, and even building, any large system.

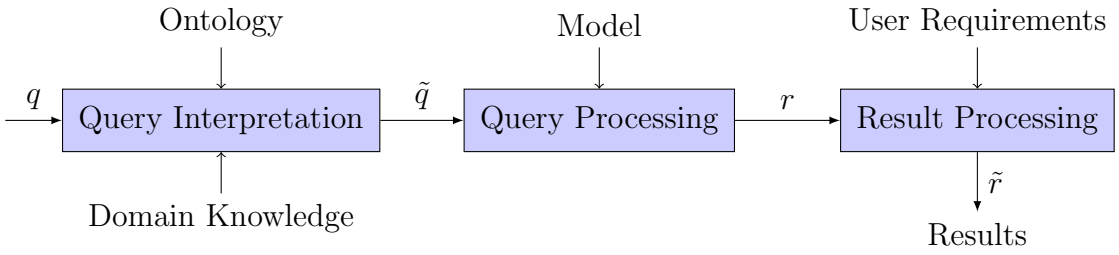

Figure 2.1: Block Diagram showing layers of our system

Figure 2.1 shows the decomposition of the overall system into three components, 1) query interpretation layer, 2) query processing layer, and 3) result processing layer.

The first layer interacts with a user, taking as input a free-keyword query, processing it and passing it to the next layer. The input query is *translated* to an internal representation suitable to be processed by the query processing layer.

The query processing layer, takes a *topic* as input, in some canonical form that is produced by the query interpretation layer. The task at this layer is to produce an intermediate result that will be then sent to the result processing layer. Typically, this intermediate result will be a ranked list of experts in the queried topic, plus some auxiliary information.

Finally, the result processing layer receives the intermediate result as input and performs operations to produce the result in the form specified by the user (or application).

2.2 Query Interpretation Layer

Because the user is not required to know the internal representations of *topics*, it could be the case that the user query does not match directly to an existing topic in the system. Therefore, in order to achieve satisfactory recall, the query interpretation layer must take steps to find out the exact intent of the user's query.

On the other hand, it is possible that the user query may map to more than one topics in the system. In this case, the query interpretation layer would need to disambiguate the user's intent in order to achieve satisfactory precision.

While the role of the query interpretation layer is no doubt important in achieving superior performance of the overall system, its working is somewhat orthogonal to the query processing layer. In this work, we shall focus on the query processing part, leaving query interpretation for future work.

2.3 Query Processing Layer

The query processing layer is at the heart of our system and the better part of this manuscript is dedicated to its discussion.

As an overview, the complete information available for searching experts is represented within a graph model and auxiliary data structures. Given a query, say, "find experts in databases", the query processing layer computes an "expertise score" for the each candidate expert, which will then be used to rank these candidates. Finally, a certain number of candidates are be deemed 'experts' and are passed on to the result processing stage.

While the issue of *coverage*, as discussed in Chapter 1, can be be considered relevant at the result processing layer, we shall try to address it in the query processing layer. We shall discuss this and other issues, along with the detailed specification of the query processing layer, in Chapter 3.

2.4 Result Processing Layer

In the simplest scenario, the query processing layer ranks the experts in the queried topic and such a ranked list is the final result desired by the user. In this case, the result processing layer's job is trivial: just forward the input to the user!

In general, the user query could be far from simple, in a manner that further filtering may be required at the result processing layer. For example, the query "find experts in databases living in India", would require extra filtering on the condition `country='India'`. This is best handled at the final layer because it allows the design of the query processing layer to be oblivious of any filtering.

Even in cases like the above, the result processing layer's task is a trivial one and we vote for reserving the following chapters solely for discussion of the query processing layer.

Chapter 3

Query Processing

Recall that we have translated the user query to a "find experts in topic T", where T is unambiguous and can be evidenced in the data. In this chapter we discuss the process of arriving at a ranked list of experts in topic T. The query processing layer has information about a number of candidate experts, possibly from a number of sources, that is used for inferring the expert-ranking. Before we proceed further, it is important to understand the internal representation of this information used by the query processing layer.

3.1 Graph model

Since the 'experts' in any expert-finding system, are typically people, a graph modelling relationships between persons is a natural way of representing the available information. For our particular case, the vertices of the graph will be authors and the edges will represent the co-author relation. The terms author, academic, and researcher, all refer to the same class of entities—persons who publish academic texts, and will be used interchangeably in the subsequent sections.

It is worth noting that there exist a number of meaningful binary relations between two authors A_1 and A_2, a few of them being:

1. A_1 and A_2 are collaborators in a scientific work (for e.g. co-authors of a paper)
2. A_1 is the academic supervisor of A_2, or vice versa
3. A_1 is influenced by A_2's work (for e.g. A_1 has cited one of A_2's paper, in a paper authored by A_1)

Clearly, the list can be populated further and building a model incorporating all such possible relations would not be feasible. Therefore, following the observation that co-authorship is perhaps the strongest[1] relation between two authors in which one of them

[1] we can assume that the supervisor-of relation is subsumed by the co-author relation

influences the other (and vice versa), we proceed further with a simple co-authorship graph.

Formally, we construct a graph $G_c = (V_a, E_c)$, with $V_a = \{A_i \mid A_i \text{ is an author}\}$, and $E_c = \{(A_i, A_j) \mid A_i, A_j \text{ are co-authors of a paper}\}$. The edges in E_c are assumed undirected. The subscripts a and c signify 'authors' and 'co-authorship', respectively.

The idea is the following: to answer the query "find experts in databases", we must score each author according to her expertise in "databases". Because scientific research tends to be of collaborative nature, a correlation between the expertise level of an author and that of her collaborators is to be expected. In determining the expertise level of an author A_i, we shall use the expertise levels of her co-authors. This forms the basis of score propagation, and will be discussed in a later section.

To provide the *seed* scores for score propagation, we extend the graph model to introduce *source* nodes from which all scores shall *originate*. The source nodes and the nature of seed scores depends on the type of information available to the system. We shall elaborate on the scoring and source nodes in the next section.

3.2 Data Sources

The DBLP[Ley, 2002] bibliography service provides information pertaining to publications in computer science. In particular, information such as: paper title, names of authors, journal/conference, and year of publication, is readily available for a large number of journal articles, conference papers, and other publications on computer science. The task of score computation will be discussed in the context of availability of DBLP data.

The DBLP dataset can be augmented by data from publicly available services such as Google Scholar[Noruzi, 2005] and Microsoft Academic Search[Roy et al., 2013]. From Google Scholar, we retrieve the number of citations of each publication enhor each conference or journal in the DBLP dataset. The impact factor of a journal (or a conference) is a measure reflecting the average number of citations to recent articles published in that journal, and is used as a representative of relative importance of the journal within its field. Journals (or conferences) with a higher impact factor are deemed to be more important than those with lower ones.

3.3 Score Computation

With the availability of our initial data as described above, we start with the following observations to construct a scoring/ranking model:

1. A higher total number of citations on an author's works should mean a higher ranking for that author

2. Higher the impact factor of the conference or journal of an author's publication, the higher its positive influence on the author's ranking

3. Having highly ranked collaborators should mean a higher ranking for that author

The above list is only a representative one and it is certainly possible to add a large number of indicators in score computation to possibly improve the accuracy of results. It is one of our goals to provide an extensible framework that can be augmented with new scoring indicators.

It is also worth mentioning that each of the above factors should be considered *ceteris paribus*, i.e. the general rule that a higher number of citations implies a higher rank, is valid if other parameters/factors are kept the same. Since the score is a composite of several *factor scores*, one way of representation would be to model it as a vector of d dimensions, where each dimension represents a single factor score. We do not adopt the vector score model, but rather combine various factor scores into a single scalar score. This choice is mainly made because it allows us to tune the amount of contribution from various independent factor scores. Scalar scores are also easier to work with.

3.3.1 Seed scores

In the three representative factors mentioned in the last section, only the last one actually utilizes the co-authorship graph structure. For the other two, a seed score can be assigned to each author, based on her number of citations and the impact factor of her publication venues.

A simple seed score assignment is as follows. For a given author, add the amount $score_p = \alpha \cdot F_C + \beta \cdot Y_p$ to her existing seed score $seed_A$, for each paper p that she has published. Here F_C represents the impact factor of the conference (or journal) of the paper p, and Y_p is the citation count of p at the current time. α and β are (possibly time varying) parameters that allow tuning of the importance of citation count and impact factor of a paper.

The parameters α and β can be functions of time to account for the desirable property that higher score be given for more recently published papers. Towards this end, we allow $\alpha := \alpha(t)$, where t is the time of publishing of the paper. And also, $\beta := \beta(t')$, where t' is the time when a citation count was incremented by 1.

$score_p$ can vary with time in potentially two ways: 1) the impact factor F_C can vary with time, and 2) the citation count Y_p can increase with time. For each increment of the citation count, an amount equal to $\beta_{t'}$ is added to the seed score of the author, where t' is the time of increment. Since the seed scores change dynamically, we use deferred updates to reduce computation costs. More on deferred update policy will be discussed in a later

section.

For a paper p with authors $A_1^p, A_2^p, ..., A_{k_p}^p$, the measure $score_p$ need not be same for all authors. If the ordering of the author names in the paper signify the relative contributions of the authors, the value $score_p$ can be divided unequally among the authors. This also maintains the property that papers with fewer number of authors contribute higher to $score_p$ of the authors than those with larger number of authors, *ceteris paribus*.

3.3.2 Score Propagation

Now that we have assigned seed scores $seed_A$ for each author A, we must update these to factor in the influence of collaborators. We supplement the co-author graph described earlier in the chapter, with edge weights signifying the strength of relation between co-authors. Then define,

$$score_A^k = \epsilon \cdot score_A^{k-1} + (1-\epsilon) \cdot \sum_{A_i \in \mathcal{N}(A)} \gamma_i \cdot score_{A_i}^{k-1}$$

$$score_A^0 = seed_A$$

$$\sum_i \gamma_i = 1$$

$$0 \leq \gamma_i \leq 1 \quad \forall i$$

$$0 \leq \epsilon \leq 1$$

where $\mathcal{N}(A)$ is the neighborhood set of A in G_c, i.e. the set of all authors that have collaborated at least once with A. γ_i are coefficients proportional to the weight of the edge (A, A_i) in G_c. The final scores for each author is $score_A^\infty$. Using the techniques similar to those presented in [Gori et al., 2007] it can be proved that the above score computation converges.

Note on Dynamic Updates

We saw in a previous section that $score_p$ can vary with time. On an update to $score_p$ for a paper p, $seed_A$ and $score_A^\infty$ for an author A of p will also need to be updated. The update in $seed_A$ can affect the score of authors other than A also, and the computation can quickly become infeasible.

To fix the above problem, it is possible to achieve lower computation costs at the expense of out-of-date scores. At each author node A, a threshold τ is maintained and if $\Delta score_A$ exceeds τ, the score propagation computation will take into account the updated $score_A$ += $\Delta score_A$ and $\Delta score_A$ is set to zero. The idea is to collect score updates in batches of τ and perform score propagation fewer times.

3.4 Coverage

Chapter 4

Conclusions and Future Work

The framework proposed in Chapters 2 and 3 provides us a good starting point towards realizing the goals of: 1) seamless integration of heterogeneous data sources, 2) ensuring good coverage in the query results, and 3) efficiently answering queries even in case of dynamically updating data.

The framework is still far from complete. A partial list of to-do tasks, in no particular order, is as follows:

1. Settle on the definition of coverage and approximate algorithms for computing the result set as desired.
2. Prove the convergence of score propagation formulation.
3. Experiment with different sets of factors that affect the scoring.
4. Design of the query interpretation layer.
5. Implementation of the complete system.

While 5 is clearly the most important to-do item and is actually the ultimate goal of this project, we believe that an approach deeply rooted in theory and design is key to developing successful systems—and such has been the theme of the work so far.

Bibliography

[Gori et al., 2007] Gori, M., Pucci, A., Roma, V., and Siena, I. (2007). Itemrank: A random-walk based scoring algorithm for recommender engines. In *IJCAI*, volume 7, pages 2766–2771.

[Ley, 2002] Ley, M. (2002). The dblp computer science bibliography: Evolution, research issues, perspectives. In *String Processing and Information Retrieval*, pages 1–10. Springer.

[Noruzi, 2005] Noruzi, A. (2005). Google scholar: The new generation of citation indexes. *Libri*, 55(4):170–180.

[Roy et al., 2013] Roy, S. B., De Cock, M., Mandava, V., Savanna, S., Dalessandro, B., Perlich, C., Cukierski, W., and Hamner, B. (2013). The microsoft academic search dataset and kdd cup 2013. In *Proceedings of the 2013 KDD cup 2013 workshop*, page 1. ACM.