Merging Ranks from Heterogeneous Internet Sources *

Luis Gravano
Computer Science Department
Stanford University
Stanford, CA 94305-9040, USA
gravano@cs.stanford.edu

Héctor García-Molina
Computer Science Department
Stanford University
Stanford, CA 94305-9040, USA
hector@cs.stanford.edu

Abstract

Many sources on the Internet and elsewhere rank the objects in query results according to how well these objects match the original query. For example, a real-estate agent might rank the available houses according to how well they match the user's preferred location and price. In this environment, "meta-brokers" usually query multiple autonomous, heterogeneous sources that might use varying result-ranking strategies. A crucial problem that a meta-broker then faces is extracting from the underlying sources the top objects for a user query according to the meta-broker's ranking function. This problem is challenging because these top objects might not be ranked high by the sources where they appear. In this paper we discuss strategies for solving this "meta-ranking" problem. In particular, we present a condition that a source must satisfy so that a meta-broker can extract the top objects for a query from the source without examining its entire contents. Not only is this condition necessary but it is also sufficient, and we show an algorithm to extract the top objects from sources that satisfy the given condition.

1 Introduction

Increasingly, sources on the Internet and elsewhere rank the objects in the results of selection queries according to how well these objects match the original condition. For such sources, query results are not flat sets of objects that match a given condition. Instead, query results are sorted starting from the top object for the query at hand.

A typical example of this kind of sources is a source that indexes text documents and answers queries using some variation of the *vector-space* model of document retrieval [1].

Example 1: Consider a World-Wide Web search engine like Excite (`http://www.excite.com`). Given a query consisting of a series of words, like "distributed databases," Excite returns the matching documents sorted according to how well they match the query. This way, Excite might return a given WWW page as the top match for the query with a *score* of 82%, some other page as the second top match with a score of 80%, and so on. ∎

Although text sources are probably the best known example, sources with multimedia objects like images are also becoming common. Matches between query values and objects in such sources are inherently "fuzzy" [2]. Even sources with more "traditional" and structured data that rank their query results are appearing on the Internet. These sources rank the highest those objects that match the user's specification the best.

Example 2: Consider a real-estate agent that accepts queries on the *Location* and *Price* attributes of the

*This material is based upon work supported by the National Science Foundation under Cooperative Agreement IRI-9411306. Funding for this cooperative agreement is also provided by DARPA, NASA, and the industrial partners of the Stanford Digital Libraries Project. Any opinions, finding, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation or the other sponsors.

Proceedings of the 23rd VLDB Conference
Athens, Greece, 1997

available houses. This agent could treat query conditions as if they were regular Boolean conditions. This way, the agent (or the user) could determine an acceptable radius around the preferred location, and an acceptable price range, and simply return all the houses with a location and price within these limits. However, there could be too many matching houses, making the user's task of going over them tedious. Also, houses with, say, a very good price but slightly outside of the acceptable location area might be missed. Therefore, some on-line real-estate agents already rank their query results (e.g., CyberHomes, at `http://www.cyberhomes.com/`). Thus, the top house returned to the user would be one that is closest to the specified location and is relatively inexpensive. As we will see, sources might choose to weigh these two criteria for their rankings in different ways. ∎

As the popularity of this type of sources increases, so does the number of *meta-brokers*. A meta-broker is a service that receives a user query, queries several relevant sources, and merges the query results into a single query result for the user that issued the query. Such meta-brokers also provide ranked query results. A key problem that a meta-broker has to address is how to extract the top matches for a query from sources that might use widely different ranking algorithms, as the following examples illustrate.

Example 3: A service like SavvySearch (`http://guaraldi.cs.colostate.edu:2000/`) queries multiple WWW search engines at once, including Excite. It then combines the results into a single ranked result. If a page p is returned only by Excite with a score of 82%, and a page p' is returned only by HotBot (`http://www.hotbot.com/`) with the same score, then both pages would be judged by SavvySearch as equally good for the query at hand. However, Excite and HotBot may use radically different scoring algorithms, so it is really not meaningful to merge the results based on the source scores. ∎

The solution is to have the meta-broker have its own scoring function that it uses to rank and merge the retrieved objects. With this scheme, each page or object retrieved is given a new *target* score, regardless of its source score, and these target scores are used to merge the results. For this to work, the meta-broker needs to retrieve enough information about the source objects to evaluate its target function on them. As we discuss in Section 4, in some cases it is not possible to retrieve all the necessary target scoring attributes, thus making it simply impossible to merge the results in a reasonable way. However, even if the meta-broker can retrieve the necessary attributes for each object, there is still the very important problem of extracting the right source objects, i.e., of extracting the source objects that will yield the highest target scores, without having to examine *all* of the source objects.

Example 4: Suppose that the score that the real-estate agent of Example 2 assigns a house for a query is $0.1 \cdot l + 0.9 \cdot p$, where l is a number between 0 and 1 that indicates how close the house is to the target location (higher values of l are better), and p is a number between 0 and 1 that indicates how close the price of the house is to the target price (higher values of p are better). Now, suppose that a meta-broker would like to weigh location and price equally, and it does so by assigning houses a score of $0.5 \cdot l + 0.5 \cdot p$.

Suppose that a user is looking for houses with preferred location in Palo Alto and a target price of $\$100K$. Furthermore, suppose that the agent has only one house in Palo Alto, with $l = 1$ (perfect location) and $p = 0.2$ (high price). *All* the remaining houses available to the agent are located in Mountain View, with $l = 0.6$ (not as good a location) and $p = 0.4$ (moderate price).

Using the definitions above, the real-estate agent would assign a score of $0.1 \cdot 1 + 0.9 \cdot 0.2 = 0.28$ to the Palo Alto house, whereas the meta-broker would assign such a house a higher score of $0.5 \cdot 1 + 0.5 \cdot 0.2 = 0.6$, since the meta-broker weighs location and price equally. Also, the agent would assign a score of $0.1 \cdot 0.6 + 0.9 \cdot 0.4 = 0.42$ to any Mountain View house, whereas the meta-broker would assign any such house a score of $0.5 \cdot 0.6 + 0.5 \cdot 0.4 = 0.5$. Consequently, the answer to the user's query from the meta-broker should be the Palo Alto house, because it has the highest score for the query according to the meta-broker's scoring algorithm. However, the real-estate agent, where the record of the Palo Alto house resides, ranks all of the other houses, which are all Mountain View houses, higher than the Palo Alto house, so the meta-broker would have to retrieve all of the agent's contents before extracting the top house, i.e., the Palo Alto house. ∎

Example 4 illustrates that it may be hard for a meta-broker to extract the best objects from autonomous sources when they use scoring functions that are different, or even slightly different, from the target function used by the meta-broker. This raises some important questions. For example, for what types of source and target scoring functions is it possible to retrieve results "efficiently," without having to retrieve full source contents? In these cases, what is the right strategy for obtaining and ranking results? For instance, given an end-user query, what types of queries, and in what order, should we submit to the sources?

Also, how much does the meta-broker need to know about the source scoring function? Turning to a negative scenario, are there "uncooperative" source scoring functions for which there is no strategy whatsoever that avoids an "exhaustive" full retrieval of the source contents?

In this paper we address these and other related questions. We start by proposing an Internet searching and ranking model (Section 2). Within this model, we then precisely characterize the classes of source and target functions that make retrieval "efficient" or "exhaustive" (Section 5). In the former case, we present an algorithm for searching sources and finding the top-ranking objects according to the meta-broker's target function (Section 3). We also describe variations to our model, and their impact on search and ranking (Section 4).

Our goal in this paper is to explore the fundamental complexity and limitations of meta-brokers. We believe that our results can guide implementors of search engines, making it clear what scoring functions may make it hard for a client meta-broker to merge information properly, and making it clear how much the meta-broker needs to know about the scoring function. This last point is important since typically search engine builders wish to keep their scoring function secret because it is one of the things that differentiates them from other sources. At the meta-broker end, we believe that our results can also be helpful in the design of the target scoring function, and in distinguishing cases where merging results is meaningful and cases where it is not.

2 Our search model

The previous section presented examples of sources and meta-brokers, and illustrated the problems that meta-brokers face when querying autonomous sources. In this section we define our searching model more precisely, and revisit the real-estate agent example in light of the new definitions.

A source S contains a single relation R_S with attributes $A_1, \ldots, A_n$. S accepts queries over R_S. A query over S simply specifies target values for some of the attributes of R_S. Thus, a query Q is an assignment of values $v_1, \ldots, v_n$ to the attributes $A_1, \ldots, A_n$ of R_S. Some of the v_i values might be *don't care* values (noted "*"). The rest of the v_i values are the *significant* values in the query.

Given a query, source S responds with the objects (i.e., tuples) of R_S that "best match" the query values [3]. The query results contain the values for $A_1, \ldots, A_n$ for every object returned. (In Section 4 we discuss sources for which this property does not hold.)

Property 1: Information in query results: *The record for an object t in the query results returned by a source S contains all the values $t[1], \ldots, t[n]$ for the attributes $A_1, \ldots, A_n$ that can be used to formulate queries over S.*

Each object t in the result for query Q is ranked according to the *source score* $Source(S, Q, t)$ that source S computes for Q and t. These scores range from 0 to 1. Since sources are autonomous, these scores could be computed in a completely arbitrary way. However, we expect them to be a function of the significant values of Q, as discussed below.

Example 5: Consider the real-estate agent S of Example 4, with relation R_S*(Location, Price)*. As mentioned above, a query to this agent may specify a target location $L =$ *Palo Alto* and some target price $P = \$100K$, for example. In other words, such a query $Q = (L, P)$ asks for houses located close to Palo Alto, and with a price not too much higher or lower than $\$100K$.

The answers that the agent gives the user are the objects of R_S ranked according to S's source score for Q [1]. This source score is arbitrary, as mentioned above. For example,

$$Source(S,(L,P),t) = \begin{cases} l & \text{if } P = * \\ p & \text{if } L = * \\ 0.1 \cdot l + 0.9 \cdot p & \text{otherwise} \end{cases}$$

where l is some number between 0 and 1 that is inversely proportional to the distance between t and the preferred location L, and p is some number between 0 and 1 that is inversely proportional to the distance between the price of t and P, as mentioned above. ■

A meta-broker receives a user query Q and returns the top objects for Q that appear in any of the available sources, according to the *target score*. The *target score* $Target(Q, t)$ for query Q and object t is some known function of the significant values in Q. The values of *Target* range from 0 to 1.

Example 5: (cont.) Continuing with the example above, we can define:

$$Target((L,P),t) = \begin{cases} l & \text{if } P = * \\ p & \text{if } L = * \\ 0.5 \cdot l + 0.5 \cdot p & \text{otherwise} \end{cases}$$

Consequently, *Target* is quite similar to *Source*: these two functions just differ in the weight that they assign to each of the two query attributes when they are both significant. ■

[1] In the remainder of the paper, we refer to both source S and its relation R_S as source S, for simplicity.

To extract the objects for a query Q with the highest *Target* scores (i.e., *the top Target objects*), a *meta-broker* queries multiple sources that hold different instances of the same relation R and that use different source score functions. The meta-broker extracts from each source S all of the objects t with $Source(S, Q, t) \geq g$, for some score $0 \leq g \leq 1$. (We will discuss how to find g in Section 3.) The meta-broker then computes the *Target* score of these objects without accessing the objects themselves, using the attribute values returned in the query results (Property 1). Finally, the meta-broker returns the top *Target* objects for the query.

Example 5: (cont.) Consider the top result that source S returns for the query Q above:

Location: Mountain View; *Price*: \$150K;
Source score: 0.42

The meta-broker can then simply discard the *Source* score for this house, and compute the *Target* score using its own algorithm. The meta-broker does this for all of the objects extracted from the sources, and returns the objects with the highest *Target* scores. ∎

The *Source* and *Target* scores for a query may vary widely, as we have seen. The following definition captures those *Source* scores that are reasonably close to a given *Target* score. This definition will be useful later to characterize the sources for which we can extract the top *Target* objects efficiently.

Definition 1: *A query Q is* manageable *at source S if there is a constant $0 \leq \epsilon < 1$ such that*

$$Source(S, Q, t) \geq Target(Q, t) - \epsilon$$

for all possible objects t. In other words, a query is manageable at a source if the Source scores for this query are not too much lower than the corresponding Target scores.

Example 6: A query Q for the real-estate agent specifying both a *Location* and a *Price* is manageable at S for the *Target* and *Source* scores defined in Example 5. In effect, we can take $\epsilon = 0.4$:

$$\begin{aligned} Target(Q, t) - \epsilon &= 0.5 \cdot l + 0.5 \cdot p - 0.4 \\ &= 0.1 \cdot l + 0.4 \cdot (l - 1) + 0.5 \cdot p \\ &\leq 0.1 \cdot l + 0.9 \cdot p = Source(S, Q, t) \end{aligned}$$

∎

Example 7: Consider the following *Target* score for the real-estate scenario:

$$Target((L, P), t) = \begin{cases} l & \text{if } P = * \\ p & \text{if } L = * \\ \max\{l, p\} & \text{otherwise} \end{cases}$$

and the following *Source* score:

$$Source(S, (L, P), t) = \begin{cases} l & \text{if } P = * \\ p & \text{if } L = * \\ \min\{l, p\} & \text{otherwise} \end{cases}$$

Then, a query Q specifying both a *Location* and a *Price* is not manageable at S, if l and p can assume arbitrary values between 0 and 1. In effect, consider an object t with $l = 1$ and $p = 0$. (Such a house has a perfect location according to the user's specification, but an exorbitant price.) Then, $Source(S, Q, t) = \min\{1, 0\} = 0 < Target(Q, t) - \epsilon = \max\{1, 0\} - \epsilon = 1 - \epsilon$, $\forall 0 \leq \epsilon < 1$. Consequently, there is no value of ϵ that will satisfy the condition in Definition 1.

Intuitively, Q is not manageable at S because top objects for *Target* can have arbitrarily low scores for *Source*. Therefore, we would have to retrieve all of the objects in S to find the top objects for *Target*, and this is exactly what we are trying to avoid. ∎

Source S is autonomous, and the meta-broker might not know S's *Source* function. However, in this section we assume that the meta-broker knows whether a query Q is manageable at S. (Section 4 relaxes this property and considers sources where it does not hold.)

Property 2: Information about source manageability: *Given a query Q and a source S, the meta-broker knows whether Q is manageable at S. Furthermore, in case it is, the meta-broker knows a value for ϵ as in the definition of manageability (Definition 1).*

Definition 2: *Let Q be a query with a significant value v_j for attribute A_j. Then, the* single-attribute query *Q_j for Q and A_j is the query that results from Q by setting the value for v_i to "*" ("don't care") for all $i \neq j$.*

To deal with sources like the one in Example 7, we introduce the notion of a *cover* for a query [2]:

Definition 3: *A set of single-attribute queries over different attributes $C = \{Q_1, \ldots, Q_m\}$ is a* cover *for a query Q if $\exists 0 \leq g_1, \ldots, g_m, G < 1$ such that $\forall$ object t:*

$$Target(Q_i, t) \leq g_i, i = 1, \ldots, m \Rightarrow Target(Q, t) \leq G$$

Intuitively, we will later use the single-attribute queries in a cover to extract a set of objects from a source that includes the top *Target* objects. This way, we will be able to work with sources at which a given query is not manageable (Example 7), or that would otherwise require potentially inefficient executions (Example 5).

[2] The notion of cover is related to that of a *complete set of atomic conditions* in [4]. (See Section 6.)

Example 8: Let Q_1 be the single-attribute query for Q and the *Location* attribute, and Q_2 be the single-attribute query for Q and the *Price* attribute. Consider the *Target* and *Source* scores of Example 5. Then, the set $\{Q_1\}$ is a cover for Q. In effect, for any $0 \le g < 1$, we can define $G = 0.5 \cdot (g+1)$. Thus, if an object t is such that $Target(Q_1, t) \le g$, then $Target(Q, t) \le 0.5 \cdot g + 0.5 \cdot p \le 0.5 \cdot (g+1) = G$. Similarly, the sets $\{Q_2\}$ and $\{Q_1, Q_2\}$ are also covers for Q. ▮

Example 9: Consider Example 7, using the min and max functions for *Source* and *Target*, respectively. The set $\{Q_1\}$ is not a cover for Q. In effect, an object t with $Target(Q_1, t) = 0$ might still have $Target(Q_2, t) = 1$, making $Target(Q, t) = \max\{0, 1\} = 1$. Therefore, for no $G < 1$ will the definition of cover hold. Similarly, $\{Q_2\}$ is not a cover for Q. However, $\{Q_1, Q_2\}$ is a cover. ▮

The main property of sources that we investigate in the rest of the paper is defined next. As we will see, if a source satisfies this property for a query, then there are cases where we do not need to extract the entire contents of the source to find the top *Target* objects for the query. Furthermore, we will show that if a source does not satisfy this property, then we always need to extract its entire contents.

Definition 4: *A source S is* tractable *for a query Q if there is a cover C for Q that consists only of queries that are manageable at S (i.e., if there is a manageable cover for Q at S, in short).*

Example 9: (cont.) Although Q is not manageable at source S, as shown above, there is a manageable cover for it, namely $\{Q_1, Q_2\}$. (Q_i is manageable at S because $Target(Q_i, t) = Source(S, Q_i, t)$ $\forall$ object t, $i = 1, 2$.) Therefore, S is tractable for Q. ▮

3 Extracting top objects from a tractable source

In this section we present an algorithm to extract the top *Target* objects for a query from a tractable source. Since we will deal with a single source, and to simplify our notation, we sometimes omit mentioning the source explicitly. For example, we use $Source(Q, t)$ as shorthand for $Source(S, Q, t)$.

Consider a query Q and a source S that is tractable for Q. The algorithm in Figure 1, which we refer to as *Top*, extracts the top *Target* objects for Q from S [3].

[3] Algorithm *Top* reduces the problem of finding the top *Target* objects for Q in S to the problem of finding all objects t in S with $Target(Q, t) > G$, for some G. [4] uses a similar strategy for processing queries over a multimedia repository.

Example 10 : Consider the real-estate agent and the scenario of Example 5. Then, Algorithm *Top* can choose $\{Q_1, Q_2\}$ as the cover for query Q (Step (1)). Since *Target* and *Source* agree on single-attribute queries, it follows that $\epsilon_1 = \epsilon_2 = 0$ (Steps (2) and (3)). We can use any $0 \le g_1, g_2 < 1$ and $G = 0.5 \cdot (g_1 + g_2)$ in the definition of cover (Definition 3). Suppose that Algorithm *Top* then picks, say, $g_1 = g_2 = 0.8$ with $G = 0.8$ (Step (4)). Then, the algorithm retrieves from S all objects t with $Source(Q_1, t) \ge 0.8$ or $Source(Q_2, t) \ge 0.8$ (Steps (5) and (6)). There is only one such house, the Palo Alto house, that matches the first condition, and no house that matches the second condition.

At this point, the algorithm has extracted all objects t with $Target(Q_1, t) \ge 0.8 + \epsilon_1 = 0.8$ or with $Target(Q_2, t) \ge 0.8 + \epsilon_2$, because Q_1 and Q_2 are manageable for S (see below). If a house t has not been retrieved, then $Target(Q_1, t) < 0.8$ and $Target(Q_2, t) < 0.8$. Because $\{Q_1, Q_2\}$ is a cover, then $Target(Q, t) \le G = 0.8$. The *Target* score for Q for the Palo Alto house is $0.6 \le 0.8$ (Step (7)), as discussed above. Consequently, the algorithm goes to Step (11) and lowers g_1 to, say, 0.7, and g_2 to, say, 0.45, assuming $\delta = 0.1$, for example.

No new objects are retrieved in Steps (5) and (6), since all of the Mountain View houses have a *Source* score for Q_1 of 0.6 ($\not\ge g_1 = 0.7$) and a *Source* score for Q_2 of 0.4 ($\not\ge g_2 = 0.45$). The Palo Alto house is retrieved again, of course. Since G for g_1 and g_2 is now 0.575, which is less than 0.6, the *Target* score for the Palo Alto house for Q, then the algorithm stops (Step (14)) and returns the object with the highest score found so far, i.e., the Palo Alto house. ▮

Theorem 1: *Let Q be a query and S a source that is tractable for Q. Then, Algorithm* Top *extracts the top Target objects for Q from S. [5]*

Consider a source S that is tractable for a query Q. We cannot guarantee that Algorithm *Top* never extracts all the objects in S. As a trivial example, consider the case when there is only one object t in S, and t is such that $Target(Q, t) = 1$. The algorithm then necessarily extracts all the objects in S, namely, object t.

Nevertheless, in many cases Algorithm *Top* is much more efficient than this. In particular, if Q has a manageable cover with high associated g_i values (Definition 3) and low associated ϵ_i values (Definition 1), then the algorithm might stop after examining just a few of the objects in S. Furthermore, as the following theorem shows, we can always define the contents of S in such a way that the algorithm stops without retrieving all of these objects from S.

Algorithm 1 *Top*
Input: A query Q and a source S that is tractable for Q.
Method:
(1) Pick a manageable cover $C = \{Q_1, \ldots, Q_m\}$ for Q at S.
(2) **for** $i = 1$ **to** m
(3) Define ϵ_i for Q_i as in Definition 1.
(4) Pick $0 \le g_1, \ldots, g_m, G < 1$ for cover C as in Definition 3.
(5) **for** $i = 1$ **to** m
(6) Retrieve all objects t with $Source(Q_i, t) \ge G_i = g_i - \epsilon_i$.
(7) Compute $Target(Q, t)$ for all objects t retrieved.
(8) **if** $\exists i$ such that $G_i \le 0$ **then**
/* We have retrieved all objects in S */
(9) Go to Step (14).
(10) **if** $\forall t$ retrieved, $Target(Q, t) \le G$ **then**
(11) Find new $0 \le g'_1, \ldots, g'_m, G' < 1$ for C
as in Definition 3 such that:
* $g'_i \le g_i \ \forall i = 1, \ldots, m$.
* $\exists j$ such that either $g'_j = 0$ or $g'_j \le g_j - \delta$, for some arbitrary, predefined constant $\delta > 0$.
(12) Replace g_i by g'_i $(i = 1, \ldots, m)$ and G by G'.
(13) Go to Step (5).
(14) Output those objects retrieved that have the highest *Target* score.

Figure 1: Algorithm to retrieve the top *Target* objects for a query from a tractable source.

Theorem 2: *Let Q be a query and S a source that is tractable for Q. Assume that there is a manageable cover $C = \{Q_1, \ldots, Q_m\}$ for Q such that $g_i - \epsilon_i > 0$ $\forall i = 1, \ldots, m$ (ϵ_i and g_i are as in Definitions 1 and 3, respectively). Then, there exist instances of S where Algorithm* Top *finds the top Target objects for Q before extracting all of the objects in S. [5]*

Theorem 2 shows that source tractability, together with the assumption in the theorem that $\forall i$, $g_i - \epsilon_i > 0$, form a *sufficient condition* for being able to sometimes extract a top *Target* object from a source without accessing all of its objects. As we will see in Section 5, source tractability is also a necessary condition: if a source is not tractable for a query, we must *always* access all of its contents to extract the top *Target* objects for the query.

4 Varying source types

Section 3 presented an algorithm to extract top objects from sources that satisfied a number of properties. However, the sources that a meta-broker has to deal with are intrinsically autonomous and heterogeneous. Some sources reveal how they process queries, while others conceal this. Some sources return quite complete information together with their query results, while others just provide quite basic data. In this section we revisit the properties of Section 3 and see in what cases we can adapt Algorithm *Top* for sources where these properties do not hold.

Property 1: Information in query results

Algorithm *Top* requires that sources return the values of the objects for those attributes with significant values in a query. In effect, Step (7) of the algorithm computes the *Target* scores for the objects retrieved using these values. However, some sources might return just object ids, or just a few of these attribute values in the query results. In such a case, a possibility for Algorithm *Top* is to access each object retrieved in its entirety to obtain all the information needed for the *Target* scores, which could be quite time consuming.

Alternatively, if the meta-broker knows how to map *Source* scores into *Target* scores for single-attribute queries (like in the real-estate agent scenario of Example 5), then it might compute the *Target* scores for the original query without accessing the actual attribute values for each object. This requires, of course, that the sources report their *Source* scores. If these scores are not available, then the meta-broker needs the attribute values.

Property 2: Information about source manageability

Algorithm *Top* requires that a meta-broker know what single-attribute queries are manageable at a source. Furthermore, a meta-broker needs to know the ϵ values (Definition 1) that bound how much lower than the *Target* scores the *Source* scores might be (Steps (2) and (3)). All this information can be derived from the *Source* scoring function of a source. Unfortunately, this function might not be publicly known, as the sources view it as their competitive advantage.

If the *Source* function for a source is not known, and Property 2 does not hold either (i.e., the meta-broker does not know whether an attribute is manageable or not, or the ϵ values), then a meta-broker can only try to guess all this information by issuing sample queries to the sources. However, whatever conclusion the meta-broker draws about a *Source* function would only be a statistical guess, since there is no way to guarantee (unless more information is available) that the corresponding source would not behave differently in the future, for example. Thus, users would still get ranked query results from the meta-broker, but they should be warned that high ranking objects might be missing from these results.

Example 11: Consider the real-estate agent of Example 5. Suppose that a meta-broker does not know whether a single-attribute query on *Location* is manageable at the source. Suppose that the meta-broker, off-line, issued a series of single-attribute queries on *Location* to the source and computed, for each such query L_i, $e_i = \max_{t \text{ retrieved}}\{\mathit{Target}(L_i, t) - \mathit{Source}(L_i, t)\}$. Based on the e_i values retrieved, the meta-broker might then decide that indeed such single-attribute queries are always manageable at the source, with associated $\epsilon = \max\{0, \max_i\{e_i\}\}$. In particular, in our real-estate scenario, ϵ would be determined to be zero, which is the right decision. ∎

To proceed as in the example above, a meta-broker needs the *Source* scores for each object retrieved. If a source does not even report these scores, then a meta-broker would have to resort to other forms of "guessing" for the ϵ values.

Other implicit properties of the source behavior

Algorithm *Top* asks sources for all objects with *Source* score G_i or higher for a single-attribute query and for arbitrary values of G_i (Steps (5) and (6)). However, a source interface might fail to allow this in several ways.

First, a source might not accept a single-attribute query for a particular attribute. For example, the real-estate agent of Example 5 might not accept queries that specify a target *Price* but not a target *Location*. In this case, we can redefine cover (Definition 3) to allow for multiple-attribute queries.

Example 12: Consider a source S and a query Q over attributes A_1, A_2, and A_3. Suppose that S does not accept single-attribute queries on A_1. However, S accepts multi-attribute query $Q_{1,2}$, which is the restriction of Q to A_1 and A_2, and S also accepts single-attribute query Q_3. Assume that $\exists 0 \leq g_{1,2}, g_3, G < 1$ such that $\forall$ object t, if $\mathit{Target}(Q_{1,2}, t) \leq g_{1,2}$ and $\mathit{Target}(Q_3, t) \leq g_3$ then $\mathit{Target}(Q, t) \leq G$. Then, $C = \{Q_{1,2}, Q_3\}$ is a cover for Q if we now allow multi-attribute queries like $Q_{1,2}$ in a cover. ∎

Thus, if we can find a manageable cover using multiple-attribute queries, then Algorithm *Top* might proceed as before. Otherwise, the meta-broker will not be able to extract the top *Target* objects from the source (Section 5).

As a second problem that a meta-broker might have with a source, the source might only return the top objects for a query, without including the *Source* scores for the objects returned. In such a case, a meta-broker does not know if it has retrieved all the objects with a *Source* score of at least G_i or not, and Step (6) needs this information. Unfortunately, the definition of manageability does not allow us to infer much about the *Source* score of an object given its *Target* score. For example, consider a source that assigns most objects a *Source* score of 1 for a given query. Then, the top k *Source* objects for that query might not include any of the top *Target* objects. Therefore, to work with such a source a meta-broker would need to know some bound on how different the *Source* and *Target* scores might be.

Finally, a source might always return a fixed maximum of, say, 200 objects per query, for efficiency reasons or to prevent users from downloading all the source's valuable contents, for example. In such a case, a meta-broker that wants all objects t with $\mathit{Source}(Q_i, t) \geq G_i$ might retrieve only those objects with $\mathit{Source}(Q_i, t) \geq G'_i$, for some higher G'_i. If these higher values (and their associated G', as in Definition 3) are not low enough to make the condition in Step (10) false, then the meta-broker cannot guarantee that it has obtained the top *Target* objects from the source, and will have to return only approximate results.

In summary, ranking objects from autonomous sources is a difficult problem. For Algorithm *Top* to work, the sources need to provide a query interface that permits "powerful enough" searches based on scores, and the sources must return "sufficient" information on the matching objects so that the meta-broker can compute its *Target* scores. Finally, the meta-broker needs to know some "fundamental properties" of the source scoring functions.

Given all that is needed by our algorithm, one may wonder if there could be some *other* algorithms that require less source functionality or less knowledge of the sources. In the next section, we show how under some very broad assumptions, essentially there is no algorithm that can rank results in a meaningful way for a source that is not tractable for a given query.

5 Source tractability as a necessary condition

In this section, we will see that if our source is not tractable, then any strategy to extract the top *Target* objects from the source using single-attribute queries must *always* retrieve all the objects. To prove this, we need to make some assumptions about *Source* and *Target* scoring functions. We believe that these assumptions are not restrictive, and all reasonable scoring functions that we can think of meet these criteria. These assumptions are in addition to the properties in Section 3.

Our first assumption about the *Source* scores for a query is that these scores can take values ranging all the way from 0 to 1. Using this assumption we rule out "constant" *Source* score functions, for example.

Assumption 1: Variability of *Source*: *Let Q be a query. Then, $\exists t_1$, t_2 objects such that $Source(Q, t_1) = 0$ and $Source(Q, t_2) = 1$.*

Our second assumption affects both the *Target* and *Source* scores for a query Q. In essence, these scores must only depend on the attributes corresponding to the significant values in Q. Thus, the attribute values for "don't care" attributes are irrelevant for *Target* and *Source*.

Assumption 2: Locality of *Source* and *Target*: *Let Q be a query and $A_1, \ldots, A_m$ the attributes with significant values in Q. Let t and t' be two objects such that $t[A_i] = t'[A_i]$ for $i = 1, \ldots, m$ (i.e., t and t' agree on all the significant attributes in Q). Then, $Target(Q, t) = Target(Q, t')$ and $Source(Q, t) = Source(Q, t')$.*

Our final assumption affects the *Target* scores for a query Q, and is related to Assumption 2. If we "improve" an object t for Q by changing its value for A_j so that it is better for Q_j, for some j, then $Target(Q, t)$ should not decrease. Also, this assumption bounds the effect of a change in $Target(Q_j, t)$ over $Target(Q, t)$.

Assumption 3: Monotonicity of *Target*: *Let Q be a query and $A_1, \ldots, A_m$ the attributes with significant values in Q. Let t and t' be two objects such that $t[A_i] = t'[A_i]$ for $i = 1, \ldots, m$, $i \neq j$ for some j. Also, $Target(Q_j, t) \geq Target(Q_j, t') - \delta$, for some $\delta \geq 0$. Then, $Target(Q, t) \geq Target(Q, t') - \delta$.*

Next, we define the class of executions for a query Q that we analyze in this section. In short, these executions follow the methodology of Algorithm *Top* in that they query the source using single-attribute queries for Q, until they have obtained "enough" objects and, hopefully, the top *Target* objects for Q. These executions decide when they have retrieved enough objects based only on the objects that they retrieve. They do not, for example, have any "magic" information about the unseen contents of the source.

Definition 5: *Let S be a source, Q a query, and $C = \{Q_1, \ldots, Q_m\}$ a set of single-attribute queries for Q. Then, a* partial retrieval *for Q and S using C is a set of objects $\{t \in S | Source(Q_i, t) > g_i$, for some $i = 1, \ldots, m\}$, with $0 < g_i < 1$, $i = 1, \ldots, m$ [4]. The g_i values are determined based on the objects retrieved, and not on the rest of the source contents.*

To prove the main result of this section, we first need the following lemma, which identifies a condition that implies manageability.

Lemma 1: *Let Q be a query and S a source for which $\exists 0 < x \leq y < 1$ such that $\forall$ object t, either $Source(Q, t) > x$ or $Target(Q, t) < y$. Then, Q is manageable at source S. [5]*

We are now ready for our main result. Consider a partial retrieval for a query Q and a source S that is not tractable for Q and that has no objects with a *Target* score of 1. The following theorem shows that such a partial retrieval might miss objects that are better than any object retrieved. In fact, we can *always* build better objects and "include" them in the source. These objects would not be retrieved, because the execution that built the partial retrieval at hand would see exactly the same top *Source* objects for each single-attribute query. Thus, this execution would stop at exactly the same point as before for each of the single-attribute queries (Definition 5), hence missing the (new) top *Target* objects. Consequently, such a partial retrieval might always be incorrect, leaving no alternative but to extract the entire source contents to obtain the top *Target* objects for Q.

Theorem 3: *Consider a query Q and a* minimal *cover $C = \{Q_1, \ldots, Q_m\}$ for Q. Assume that $\exists j$ such that Q_j is not manageable at source S, and Q_i is manageable at source S, $\forall i \neq j$. Consider a partial retrieval for Q and S using C, and let $G = \max_{t \text{ retrieved}}\{Target(Q, t)\}$. Assume that $G < 1$.*

[4] This definition excludes executions that request all objects with a non-zero *Source* score for Q_i, since g_i has to be greater than zero. However, this is not a limitation for most sources, where *Source* scores have finite precision.

Then, we can build an object l not in the partial retrieval such that Target$(Q, l) > G$.

Proof: Let $0 < g_i < 1$, $i = 1, \ldots, m$, be the values used by the partial retrieval for Q and S using C (Definition 5). For every $i \neq j$, pick an object t_i such that $Source(Q_i, t_i) \leq g_i$. (Such objects exist from Assumption 1.) From the choice of t_i and the definition of partial retrieval, it follows that t_i is not retrieved by query Q_i. Let $a_i = Target(Q_i, t_i)$ $(0 \leq a_i \leq 1)$.

From the minimality of C it follows that $C - \{Q_j\}$ is not a cover for Q. Then, there is an object l_0 such that $Target(Q_i, l_0) \leq a_i$ $\forall i \neq j$ and $Target(Q, l_0) > G$. Otherwise, $C - \{Q_j\}$ would be a cover for Q. (If $m = 1$, just pick any object l_0 with $Target(Q, l_0) > G$.) Furthermore, $Target(Q_i, l_0) \leq a_i = Target(Q_i, t_i)$ $\forall i \neq j$.

We now build an object l_1 using the t_is and l_0:

$$l_1[i] = \begin{cases} t_i[i] & \text{if } i = 1, \ldots, m,\ i \neq j \\ l_0[i] & \text{otherwise} \end{cases}$$

From the choice of l_1 it follows that:

- $i = 1, \ldots, m$, $i \neq j$: $Target(Q_i, l_1) = Target(Q_i, t_i)$, because $l_1[i] = t_i[i]$ and using Assumption 2. Furthermore, $Target(Q_i, t_i) = a_i \geq Target(Q_i, l_0)$.
- Otherwise: $Target(Q_i, l_1) = Target(Q_i, l_0)$, because $l_1[i] = l_0[i]$ and using Assumption 2.

Then, $Target(Q_i, l_1) \geq Target(Q_i, l_0)$, $\forall i$. Hence, from Assumption 3, it follows that $Target(Q, l_1) \geq Target(Q, l_0) > G$. Also, for $i = 1, \ldots, m$, $i \neq j$, $Source(Q_i, l_1) = Source(Q_i, t_i) \leq g_i$. Hence l_1 is not retrieved by any of the Q_i queries, $i \neq j$.

Next, we build another object l_2. We will use l_1 and l_2 to construct the final object l that we need for our proof. Let $0 < \delta < Target(Q, l_1) - G$. Now, let $x = g_j$ and $y = \max\{x, Target(Q_j, l_1) - \delta\}$. (Then, $0 < x \leq y < 1$.) Since Q_j is not manageable at S, from Lemma 1 it follows that there is an object l_2 such that $Source(Q_j, l_2) \leq x$ and $Target(Q_j, l_2) \geq y$. Then, $Source(Q_j, l_2) \leq g_j$ and $Target(Q_j, l_2) \geq Target(Q_j, l_1) - \delta$.

Finally, let us define object l by letting $l[i] = l_1[i]$ $\forall i \neq j$ and $l[j] = l_2[j]$. Then,

- $i \neq j$: $Target(Q_i, l) = Target(Q_i, l_1)$.
- Otherwise: $Target(Q_j, l) = Target(Q_j, l_2) \geq Target(Q_j, l_1) - \delta$.

Then, from Assumption 3 it follows that $Target(Q, l) \geq Target(Q, l_1) - \delta > Target(Q, l_1) - Target(Q, l_1) + G = G$. Also,

- $i = 1, \ldots, m$, $i \neq j$: $Source(Q_i, l) = Source(Q_i, l_1) \leq g_i$.
- Otherwise: $Source(Q_j, l) = Source(Q_j, l_2) \leq g_j$.

Thus, we have constructed an object l that satisfies the conditions in the theorem. ∎

Corollary 1: *Let $C = \{Q_1, \ldots, Q_m\}$ be a (not necessarily minimal) cover for the query Q of Theorem 3 such that it does not contain any manageable cover for Q. Then, we can still build an object l as in Theorem 3 for any partial retrieval for Q and S using C. [5]*

Note that the main results of this section only cover algorithms that work via multiple single-attribute queries. We believe that this is not a restriction for most sources, since we expect the *Source* scores to match the *Target* scores for single-attribute queries more often than for multi-attribute queries.

6 Related work

The problem of merging document ranks from multiple sources has received recent attention in the information retrieval field, where it is often referred to as the *collection fusion* problem. Given a query, the goal is to extract as many of the *relevant* documents as possible from the underlying document collections. As with our problem, key decisions include how far "down" each document rank to explore, and how to translate *Source* scores (*local* similarity measures) into *Target* scores (usually *global* similarity measures). An approach to address these problems is to learn from the results of training queries [6]. Another approach is to calibrate the document scores from each collection using statistics about the word distribution in the collections [7]. One important difference between this line of work and ours is that we want to *guarantee* that meta-brokers extract the top *Target* objects from the sources and return these objects ordered according to their *Target* scores. In contrast, the work on the collection fusion problem develops *heuristics* or techniques for placing relevant documents (a subjective notion) as high as possible in the combined document ranks for a query, sometimes using the *Source* scores as indicators of relevance.

For document collections, it is particularly hard to compute the *Target* score for a document from the query results that are typically returned by text search engines. In effect, these results do not include entire documents, and have very little information other than the *Source* scores. To address this problem, the STARTS protocol proposal [8] developed at Stanford specifies what information should accompany the query results that a text search engine returns so that document rank merging is facilitated.

A closely related problem is how to query a repository of complex, multimedia objects. These objects might have attributes like images and text. Thus, the matches between query values and such multimedia attributes are inherently fuzzy, and the objects are ranked according to how well they match the query values. The work in [3] and [4] studies how to query such repositories efficiently. In particular, [3] studies upper and lower bounds on the number of objects that we need to extract from a repository so that the overall top objects are retrieved and returned to the user that issued a query. [4] addresses the cost-based optimization of queries over such repositories. This work assumes that a single repository handles all attributes of an object. Therefore, there is no need to "calibrate" the scores that an object gets for a particular attribute, for example. Using our terminology, all single-attribute queries are manageable with $\epsilon = 0$. (See Section 7 for further discussion.)

Finally, there has been a significant amount of work on querying multiple heterogeneous sources. In this paper, we assume that all sources export a uniform interface so they can all answer queries over the same set of attributes. We can use the techniques in [9, 10], for example, to build *wrappers* around the sources and provide the illusion of such a uniform interface.

7 Conclusion

Many sources rank the objects in query results according to how well these objects match the original query. In this environment, meta-brokers usually query multiple autonomous, heterogeneous sources that might use varying result-ranking strategies. In this paper we have studied two crucial problems that a meta-broker faces: guaranteeing that it has extracted all the top objects for a user query from the underlying sources, and re-ranking these objects according to its own criterion. These are difficult problems, and the goal of this paper is to characterize the sources where we have some hope of dealing with these problems efficiently. We have presented necessary properties that any source should satisfy, under broad assumptions. If a source does not verify these properties, then a meta-broker might miss top objects from the source, unless all of the source's contents are retrieved. We have also described a simple algorithm to extract the top objects from a source where our properties hold.

The results in this paper, and Algorithm *Top* in particular, do not guarantee efficient executions. In effect, Algorithm *Top* might retrieve large portions of a source when searching for top *Target* objects. An interesting open issue is then the optimization of queries over multiple sources, perhaps using statistics on the sources' contents to obtain small ϵ_i and large g_i values, for example. A promising direction is to adapt the work in [3] and [4] to our distributed, heterogeneous scenario. Another interesting issue is how to deal with sources that do not satisfy the properties and assumptions that our results need. We touched on this issue in Section 4, but we need to explore further, for example, how to deal with sources that return no more than, say, 200 objects per query. These characteristics also impact the optimization of queries over these sources.

References

[1] Gerard Salton. *Automatic text processing: the transformation, analysis, and retrieval of information by computer.* Addison Wesley, 1989.

[2] W. Niblack, R. Barber, W. Equitz, M. Flickner, E. Glasman, D. Petkovic, P. Yanker, and C. Faloutsos. The QBIC project: Querying images by content using color, texture, and shape. In *Storage and retrieval for image and video databases (SPIE)*, pages 173–187, February 1993.

[3] Ronald Fagin. Combining fuzzy information from multiple systems. In 15^{th} *ACM Symposium on Principles of Database Systems*, June 1996.

[4] Surajit Chaudhuri and Luis Gravano. Optimizing queries over multimedia repositories. In *Proceedings of the 1996 ACM SIGMOD Conference*, 1996.

[5] Luis Gravano and Héctor García-Molina. Merging ranks from heterogeneous Internet sources. Technical Report SIDL-WP-1997-0063, Stanford University, February 1997. Accessible as `http://www-diglib.-stanford.edu/cgi-bin/WP/get/SIDL-WP-1997-0063`.

[6] Ellen M. Voorhees, Narendra K. Gupta, and Ben Johnson-Laird. The collection fusion problem. In *Proceedings of the* 3^{rd} *Text Retrieval Conference (TREC-3)*, 1995.

[7] James P. Callan, Zhihong Lu, and W. Bruce Croft. Searching distributed collections with inference networks. In *Proceedings of the* 18^{th} *Annual SIGIR Conference*, 1995.

[8] Luis Gravano, Chen-Chuan K. Chang, Héctor García-Molina, and Andreas Paepcke. STARTS: Stanford proposal for Internet meta-searching. In *Proceedings of the 1997 ACM SIGMOD Conference*, May 1997.

[9] J.-C. Franchitti and R. King. Amalgame: a tool for creating interoperating persistent, heterogeneous components. In *Advanced Database Systems*, pages 313–36. Springer-Verlag, 1993.

[10] Yannis Papakonstantinou, Hector Garcia-Molina, Ashish Gupta, and Jeffrey Ullman. A query translation scheme for rapid implementation of wrappers. In *Fourth International Conference on Deductive and Object-Oriented Databases*, pages 161–186, National University of Singapore(NUS), Singapore, 1995.