Profile-based Retrieval of Records in Medical Databases

Anastasios Kementsietsidis, Ph.D. Lipyeow Lim, Ph.D. Min Wang, Ph.D.
IBM T.J. Watson Research Center, 19 Skyline Drive, Hawthorne, NY 10532, USA.

Ontologies establish relationships between different terms, yet their potential in querying has not yet been fully realized. In this paper, we study the problem of ontology-supported profile-based retrieval of medical records. We present an algorithm that provides two independent techniques (used in isolation or in unison) to address the shortcomings of existing keyword-based retrieval solutions, and provide an implementation and experiments to illustrate the merits of our approach.

1. Introduction

Coding systems such as SNOMED or ICD9, and ontologies like the National Cancer Institute (NCI) Thesaurus, encode medical terms and their relationships, and provide a common basis on which to describe and exchange medical data (e.g., Electronic Medical Records (EMR)). Since the majority of such data are stored in relational databases, it becomes imperative to study the *interaction* between ontologies and relationally-stored data. Indeed, our previous work studied how to support ontology-based keyword searches over medical databases [5]. However, keyword search is just one simple application that results in from this interaction, and more complex applications are possible, as the following examples illustrate.

Medical researchers interested in conducting a clinical trial use a profile (a.k.a. *inclusion criteria*) to describe patients that can participate in the trial. In its simplest form, this profile contains a set of conditions the researchers deem relevant to the trial and that the participating patients must have. There are almost 70,000 real examples of such profiles in the *Clinical Trials* website [2]. Here, consider a profile with three conditions, namely, c_1, c_2, and c_3. Given the profile and a patient database, the medical researchers would like to (a) *filter* the database and identify an initial pool of candidate patients whose medical history matches (or is close) to the profile; and (b) carefully screen the pool of candidates and identify the most promising ones. While step (b) requires human expertise, it is imperative to provide some form of automation for step (a) since, given the usually large number of database records, manually finding the initial pool of candidate patients is extremely time consuming.

A naive solution to automate step (a) is to use ontology-based keyword searches [5]. For our example, we can evaluate three keyword searches (one per condition), and retrieve three patient lists, namely, $\mathcal{L}_{c_1}$, $\mathcal{L}_{c_2}$, and $\mathcal{L}_{c_3}$. Then, the intersection of these lists constitutes our initial pool of candidates since it includes patients having all the three conditions. Depending on the number of patients in the intersection, this naive solution might be satisfactory, especially in simple settings with *small* sets of *carefully selected* conditions. If larger sets of conditions are selected, or if the conditions are not carefully selected, then such an intersection often results in the empty set (no patient satisfies all the selection criteria). At this point, the naive solution fails since it is up to the researchers to manually consider intersections between subsets of the initial set of lists. For example, they may consider intersecting only lists $\mathcal{L}_{c_1}$ and $\mathcal{L}_{c_2}$ to only find patients with the c_1 and c_2 conditions, or only lists $\mathcal{L}_{c_2}$ and $\mathcal{L}_{c_3}$, or in certain settings relax completely the requirements and consider the *union* of all the lists, which corresponds to the list of patients with at least one of the conditions (note that this union usually results in a very large pool of candidates). This manual experimentation is clearly undesirable since in the worst case the combinations of lists that must be considered is *exponential* in the size of the profile. The situation is exacerbated in settings when the medical researchers determine that no combination of lists results in a satisfactory pool of candidates. Then, a new profile of *similar* conditions should be selected and the whole process must be repeated.

The objective of this paper is to introduce an automated procedure that addresses the shortcomings of the naive procedure, and can be used when the latter produces unsatisfactory results. In more detail, we present an algorithm that automatically interprets the profile conditions using the related ontology in order to return a non-empty set of candidate patients. In a nutshell, our algorithm interprets the profile conditions along two independent dimensions. First, it identifies *incompatibilities* between the conditions in the profile, where these incompatibilities are manifested by having either a very small number or no common patients at all in the intersecting patient lists corresponding to

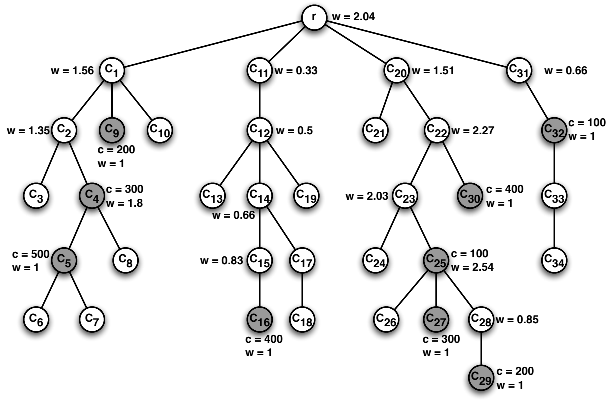

Figure 1: A sample ontology $\mathcal{O}$ and marked profile $\mathcal{P}$

visit_ID	patient	condition
1	John	Brain Neoplasm
2	Jim	Arthritis
3	Tim	Brain Tumor
4	Jim	Colon Neoplasm

Figure 2: A sample relation $\mathcal{R}$

the individual profile conditions. Once identified, the algorithm proceeds to remove from the profile the conditions that cause the incompatibilities, thus relaxing the set of conditions considered. Intuitively, this relaxation has an effect on the *recall* of the algorithm, as we discuss in more detail later on. Along the second dimension, the algorithm augments the profile conditions with more general conditions that are semantically related with the conditions initially found in the profile. The addition of these more general conditions is useful when the initial set of conditions are very specific and result in limited pools of candidate patients. Intuitively, our latter relaxation has an effect on the *precision* of the algorithm, as the following section illustrates. Both techniques are possible due to the use of medical ontologies and in the following paragraphs we present in more detail how these are achieved.

Although we use an example from clinical trials, the above functionality is equally important in other settings. For example, drug companies interested in targetted advertising would like to identify the most promising people to which to market specific drugs. Once more, this requires the ability to match the conditions for which drugs are prescribed to the conditions of the patients. Finally, doctors might also find the above functionality useful. A doctor who has a patient with a certain set of conditions might want to see what is the prognosis for patients with similar profiles. Our system can also be useful there.

2. A Profiled-based Retrieval Algorithm

In what follows, we describe the algorithm to support profile-based retrieval of medical records. The algorithm (shown in Figure 3) accepts as input a relation $\mathcal{R}$ (e.g., a set of patient records like those in Figure 2), an ontology $\mathcal{O}$ (e.g., the NCI thesaurus, or the sample ontology in Figure 1), and a profile $\mathcal{P}$ (e.g., the set of conditions corresponding to the grey nodes in Figure 1), and returns as output a list $\mathcal{L}$ of patients whose medical histories in $\mathcal{R}$ semantically match profile $\mathcal{P}$. The algorithm has four distinct steps which we review in detail next. We assume that the input ontology $\mathcal{O}$ is a *directed acyclic graph (DAG)*. This is the case for common ontologies like the NCI Thesaurus used in our experiments. However, we present here the case where the input ontology is a tree. We do this both due to lack of space and for ease of presentation since the algorithm for DAGs is much more involved. We discuss some of the main differences between trees and DAGs at the end of the section.

```
    input  : Relation $\mathcal{R}$, Ontology $\mathcal{O}$, Profile $\mathcal{P}$
    output: A list $\mathcal{L}$ of candidates from $\mathcal{R}$
    // Step 1:  Preprocessing
1   foreach node (condition) $c_i$ in a depth-first traversal of $\mathcal{O}$ do
2       $card_{LOC}(c_i)$ = tuples in $\mathcal{R}$ with condition $c_i$;
    // Step 2:  Incompatible condition detection
3   foreach node $c_i \in \mathcal{O}$ where $c_i$ is also in $\mathcal{P}$ do
4       path $(c_i, r)$ = path of nodes from $c_i$ to the root $r$ of $\mathcal{O}$;
5       foreach $c_j$ in path $(c_i, r)$ do
6           increment weight $(c_j)$ by $\frac{|\text{path}(c_j,r)|}{|\text{path}(c_i,r)|}$;
7   for $k = 1$ to $|\mathcal{O}|$ do
8       $\mathcal{P}_X$ = top-$k$ nodes of $\mathcal{O}$, sorted by weight;
9       density = $\frac{\sum_{c_i \in (\mathcal{P} \cap \mathcal{P}_X)} card_{LOC}(c_i)}{\sum_{c_i \in \mathcal{P}} card_{LOC}(c_i)}$;
10      if density $\geq$ thr$_{\text{incomp}}$ then break;
    // Step 3:  Condition generalization
11  foreach node $c_i$ in a depth-first, post-order, traversal of $\mathcal{O}$ do
12      if $c_i$ is a leaf-node then $card_{SUB}(c_i)$ = $card_{LOC}(c_i)$;
13      else
14          foreach child node $c_j$ of $c_i$ do
15              if weight$(c_j) > 0$ then
16                  add $card_{SAT}(c_j)$ to $card_{SAT}(c_i)$;
17              add $card_{SUB}(c_j)$ to $card_{SUB}(c_i)$;
18      if weight$(c_i) > 0$ then generic $(c_i)$ = $\frac{card_{SAT}(c_i)}{card_{SUB}(c_i)}$;
19      else generic $(c_i)$ = 0;
20  foreach pair $(c_i, c_j)$ of nodes in $\mathcal{P}_X$ do
21      if there is no node $c_l$ in $\mathcal{P}_X$, with $c_l \in$ path$(c_i, c_j)$ then
22          add edge $(c_i, c_j)$ in $\mathcal{O}_k$
23  foreach sub-tree $\mathcal{T}$ of $\mathcal{O}_k$ do
24      foreach node $c_i$ in a depth first, pre-order, traversal of $\mathcal{T}$ do
25          if generic$(c_i) \geq$ thr$_{\text{gen}}$ then add $c_i$ to $\mathcal{P}_F$;
26          else push child nodes of $c_i$ to visiting stack;
27          pop visiting stack;
    // Step 4:  Result computation
28  foreach node $c_i$ in $\mathcal{P}_F$ do
29      res$_{c_i}$ = tuples in $\mathcal{R}$ with a sub-condition of $c_i$;
30  $\mathcal{L} = \cap_{c_i \in \mathcal{P}_F}$ getid(res$_{c_i}$);
31  return $\mathcal{L}$;
```

Figure 3: The algorithm

Preprocessing step: During this step, the algorithm performs a simple depth-first traversal of the ontology $\mathcal{O}$, and annotates each node c_i in the ontology with the cardinality of the set of tuples in $\mathcal{R}$ whose condition is c_i. To keep Figure 1 simple, we only show these cardinalities along each condition of profile $\mathcal{P}$.

Incompatible condition detection step: Intuitively, one reason the naive procedure fails is when profile conditions are *incompatible*, that is, a few or no patients in $\mathcal{R}$ have a medical history that matches all of these conditions. The users are unaware as to which of the conditions in the profile are the source of this incompatibility. Without some form of automation, the users might have to manually experiment with subsets of the conditions in the profile to find a sizeable pool of candidate patients. Since there is an exponential number of subsets (for a profile $\mathcal{P}$ there are $2^{|\mathcal{P}|}$ such subsets) this manual experimentation is time consuming. Even worse, the users have no guidance as to how to systematically consider and find the *most promising* subsets of conditions. The objective of this step is to provide an automated procedure to detect the conditions in the profile that are potentially incompatible with the remaining profile conditions. In a nutshell, the algorithm uses the ontology $\mathcal{O}$ to determine the semantic relationship between the various conditions in $\mathcal{P}$. Then, the algorithm uses these semantic relationships to identify *clusters* of conditions that are closely related semantically (according to $\mathcal{O}$) and, at the same time, identify potential condition *outliers* (again, according to $\mathcal{O}$), that is, conditions which are not closely related to the other profile conditions. As such, these outlier conditions can be candidates for exclusion.

To determine condition clusters and outliers, the algorithm relies on the notion of *weighted path sharing*. In more detail, for a node $c_i \in \mathcal{O} \cap \mathcal{P}$, consider the path (in the generalization hierarchy of $\mathcal{O}$) of conditions from c_i to the root r of $\mathcal{O}$ (denoted by $path(c_i, r)$). This path consists of conditions that are related to c_i since they semantically subsume it. Then, the premise of *weighted path sharing* is that two conditions c_j and c_l are semantically related, and are part of the same cluster, if their corresponding paths overlap *considerably*. To quantify this overlap, we assign weights to nodes of the ontology, with a node c_i in $\mathcal{O} \cap \mathcal{P}$ being initially assigned a weight of one, while the weight of every node c_j in the path from c_i to the root r is incremented by $\frac{|path(c_j,r)|}{|path(c_i,r)|}$ (notice that this increment is monotonically decreasing, the further c_j is from c_i). It is not hard to see that for a node $c_j \in \mathcal{O}$, this weight assignment results in weights whose value is proportional to (a) the number of profile conditions that are present under the same subtree rooted at c_j; and (b) the depth of these conditions, relatively to c_j. So for example, in Figure 1, node c_{22} has a (high) weight of 2.27 since four from the nine conditions in $\mathcal{P}$ are in the subtree rooted at node c_{22}. This weight is higher than that of node c_{20}, since the profile conditions are closer (depth-wise) to c_{22} than to c_{20}, and therefore the conditions are semantically more closely related with the former node than with the latter. Also notice that nodes with few conditions in their subtrees, like c_{12}, indeed have low weights when compared to nodes like c_{22}. Using weights, we identify clusters of conditions from $\mathcal{P}$ by locating subtrees of $\mathcal{O}$ rooted in nodes with high weights, while subtrees rooted in nodes with low weights indicate potential condition outliers.

Once clusters and outliers are identified, we must decide the level of relaxation, that is, how many of the outliers should be ignored while relaxing the profile $\mathcal{P}$. We allow the user to provide some input at this stage, through a threshold parameter, called $\text{thr}_{\text{incomp}}$. By relaxing the profile a number of patient records (those corresponding to ignored condition outliers) will not be considered in later stages of the algorithm. Through the threshold parameter, the user essentially gives an approximation of the percentage of these ignored records she is willing to tolerate, when compared to the full set of records retrieved when no relaxation occurs. Intuitively, our algorithm allows the user to provide a measure of *recall* (the ratio of patients considered after the relaxation over the original set of patients considered by $\mathcal{P}$), and since this parameter is user-specified, our algorithm can provide any level of relaxation the user desires. Given this parameter, the algorithm automatically determines which outliers should be ignored to achieve the required level of relaxation and stores in profile $\mathcal{P}_X$ the conditions of $\mathcal{P}$ in the selected clusters. As an example, for the profile of Figure 1 and for $\text{thr}_{\text{incomp}} = 0.8$, the extended profile $\mathcal{P}_X$ will not include the outliers c_{16} and c_{32}, while for $\text{thr}_{\text{incomp}} = 0.9$ only the outlier c_{32} is not included.

Condition generalization step: Another reason why the naive procedure might fail is that the users consider conditions in the profile $\mathcal{P}$ for which there are few or no patients in $\mathcal{R}$ with *exactly* these conditions. However, relation $\mathcal{R}$ might include a sizable population of patients that have been identified as having a slightly more generic condition than the one in the profile. These patients might still be relatively good candidates for the trials. Although we deal with expert users, determining which conditions in the profile must be generalized, at which level, and by which *precise* condition, is a demanding activity that requires both in-depth knowledge of medical ontologies (e.g. the NCI thesaurus), and in-depth understanding of the characteristics of the particular relation $\mathcal{R}$ used to create a candidate pool. The objective of this step is to automatically compute appropriate condition generalizations that are tuned to the provided ontology $\mathcal{O}$ and relation $\mathcal{R}$. The only user input required is a threshold indicating the desired generalization level.

In more detail, the first aspect of condition generalization, called *hyponym generalization*, is that the algorithm considers, for each condition c, not only the patients in $\mathcal{R}$ with this *exact* condition but also all the patients in $\mathcal{R}$ that have a condition c' which is a subcondition of c, according to the ontology $\mathcal{O}$. The second aspect of condition generalization, called *hypernym generalization* is that the algorithm determines the conditions that appear in profile $\mathcal{P}$ that have few or no patients in $\mathcal{R}$, and it *replaces* these conditions with more generic conditions that appear in the ontology $\mathcal{O}$ and result in a more sizeable pool of candidates from $\mathcal{R}$. Since both aspects of condition generalization result in patients that wouldn't have been considered with the initial profile, the users must have some control as to the level of this relaxation. The threshold parameter thr_{gen} is used for this purpose. Intuitively, through this parameter, the user has complete control over the *precision* (the ratio of the patients retrieved based on profile $\mathcal{P}$ over the total number of patients retrieved due to the relaxation) of the algorithm and therefore she can decide, depending on the results of the relaxation, to increase or decrease the level of relaxation in future runs of the algorithm.

On the technical side, the algorithm performs appropriate generalizations by considering in turn each subtree of $\mathcal{O}$ corresponding to a condition cluster (computed in the previous step). For each such sub-tree, it determines (through a depth-first traversal) the set of conditions that: (a) each condition in the set satisfies the relaxation constrain enforced by the thr_{gen} parameter; (b) no two conditions in the set are such that one is a generalization of the other; and (c) for each condition in the set, any further generalization violates either condition (a) or condition (b) above, and therefore the conditions in the set are the most generalized possible. This set of conditions creates a new profile called $\mathcal{P}_F$.

Result computation step: The last step considers the intersection of patients that satisfy all the conditions in profile $\mathcal{P}_F$ whose identifiers are returned to the user.

Discussion: An interesting aspect of our algorithm is that the relaxations in steps 2 and 3 are independent of each other. To see this, assume that we terminate the algorithm after the end of step 2. Then, we can use the result of this step to remove any outlier conditions from profile $\mathcal{P}$ and consider only patients whose medical histories match exactly the remaining conditions (without any generalizations). On the other hand, we can consider the whole profile $\mathcal{P}$ (with its possible ourliers) and use step 3 of the algorithm to find generalizations for the conditions there. The above illustrate not only the flexibility of our algorithm but also its ability to combine complementary techniques.

For simplicity in our presentation, we considered tree ontologies. However, our implementation actually supports directed acyclic graphs (DAG)s. Notable complications to support DAGs become apparent when one considers the computation of node weights, node (card_{LOC}) and sub-tree (card_{SUB}, card_{SAT}) cardinalities, and the computation of sub-structures for $\mathcal{O}_k$. As an example, in the case of node weights, special care must be taken so that the weight of a node c_i is propagated along all the paths from c_i to the root r of $\mathcal{O}$. Node c_i might appear in a DAG at different depths. A node c_j might appear at different points of multiple paths (of different lengths) from c_i to r. Then, the weight of c_j must be adjusted by its (different) weight along each of these paths.

3. Implementation

Our algorithm is implemented using the C++ programming language. For our experiments, we used the NCI thesaurus from which we extracted conditions and generated profiles. In terms of patient records, due to privacy reasons we had no access to real records. Therefore, we synthetically generated records but, as expected, these records cannot accurately reflect the medical profile of a real person since the generation involves randomization.

In the following paragraphs, we present some of the key findings of our experimentation. We start by considering the profile $\mathcal{P}$ shown in Figure 4. We used this profile as input to our algorithm, along with the NCI thesaurus and one million synthetically generated patient records. The threshold parameters $\text{thr}_{\text{incomp}}$ and thr_{gen} were both set to 1, i.e., we wanted to consider all the conditions and avoid any generalizations that might reduce its precision. Interestingly enough, in spite of the severe limitations imposed by the thresholds, our algorithm retrieved patients whose conditions match the profile in Figure 5. In more detail, the algorithm determined that the conditions of *"Skin Angiosarcoma"* and *"Skin Kaposi s Sarcoma"* are exactly subsumed by the condition of *"Malignant Dermal Neoplasm"* without compromising the precision requirement of the thr_{gen} threshold. So, instead of executing two independent queries for the former two conditions the algorithm decided instead to execute a query using the latter. Similarly, the algorithm determined that condition *"Cherry Hemangioma"* is subsumed by condition *"Hemangioma of the Skin"* (also in the profile) which, in turn, is subsumed by *"Benign Cutaneous Vascular Neoplasm"*. Once more, without sacrificing precision, three conditions are replaced by a single equivalent one, illustrating that the algorithm can perform *compression* of the profile and avoid executing unnecessary queries. As expected, given the large number of conditions in $\mathcal{P}$, this experiment found

Malignant Cutaneous Vascular Neoplasm	
Malignant Dermal Neoplasm	
Benign Cutaneous Fibrous Neoplasm	
Benign Cutaneous Vascular Neoplasm	
Cutaneous Fibrous Tissue Neoplasm	Pure Cutaneous Mastocytosis
Cutaneous Granular Cell Tumor	Skin Angiosarcoma
Cutaneous Leiomyoma	Skin Hemangioendothelioma
Cutaneous Lipomatous Neoplasm	Skin Kaposi s Sarcoma
Cutaneous Mastocytosis	Hemangioma of the Skin
Cutaneous Vascular Neoplasm	Cherry Hemangioma

Figure 4: The profile $\mathcal{P}$

Malignant Dermal Neoplasm	
Benign Cutaneous Vascular Neoplasm	
Cutaneous Lipomatous Neoplasm	Cutaneous Vascular Neoplasm
Cutaneous Granular Cell Tumor	Cutaneous Mastocytosis
Cutaneous Fibrous Tissue Neoplasm	Cutaneous Leiomyoma

Figure 5: The profile $\mathcal{P}_F$

no synthetic patients satisfying all conditions.

Next, we used again the profile of Figure 4 but set $\text{thr}_{\text{incomp}}$ to 0.8. This resulted in a subset of the profile in Figure 5 that does not include the *"Cutaneous Lipomatous Neoplasm"*, *"Cutaneous Granular Cell Tumor"* and *"Cutaneous Fibrous Tissue Neoplasm"* conditions. Correctly reacting to the relaxation, our algorithm indeed dropped from consideration the conditions that retrieve approximately 20% of the tuples. At the same time, this relaxation resulted in 19 synthetic patients that satisfy the remaining conditions.

We also studied the independent effects of the thr_{gen} parameter by setting it to 0.45 while setting $\text{thr}_{\text{incomp}}$ back to one. Then profile $\mathcal{P}$ resulted in a profile $\mathcal{P}_F$ consisting of the conditions *"Dermal Neoplasm"*, *"Benign Dermal Neoplasm"* and *"Malignant Dermal Neoplasm"*. From these three conditions, the first essentially generalizes the *"Cutaneous"*-prefixed conditions in $\mathcal{P}$, the second generalizes the *"Benign Cutaneous Vascular Neoplasm"* conditions (and its subconditions mentioned earlier), while the last condition is in $\mathcal{P}$ and is not further generalized.

Certain points need further clarifications. First, one might wonder how exactly we picked the threshold values used in our experiments and what guidance, if any, we can offer to the user to select appropriate thresholds. Clearly, having the user try different values unaided is unsatisfactory. Therefore, our algorithm apart from returning to the user the list of candidate patients, it also outputs (in a user-friendly manner) a number of intermediate structures, like the sets $\mathcal{P}_X$ and $\mathcal{P}_F$, the sub-trees $\mathcal{T}$ of $\mathcal{O}_k$, and more importantly, for each of the nodes in these structures it outputs their corresponding weights. After an initial run of the algorithm, possibly using the value one for both thresholds (as in our experiments), the user can inspect these intermediate structures. By inspecting node weights, she can decide how to revise these thresholds in subsequent algorithm runs. It is expected that after only a small number of runs, the user will be satisfied with the results of the chosen thresholds. Even if a larger number of runs is deemed necessary, each run requires a very short time (with input the NCI thesaurus of approximately 65,000 concepts, and one million patient records, each algorithm run requires less than 10 seconds on a commodity desktop).

Finally, we conclude by noting that in the profile used here most of the conditions were closely related (they formed one loose cluster). We also experimented by including condition outliers in our profiles, for example, including the condition *"Dermatitis"* in the profile of Figure 4. These outliers were easily identified by our algorithm and where excluded from consideration while retrieving patient records.

4. Related work

While our work considers relaxations of input keywords, keyword search in relational databases retrieves records that mention *all* input keywords (e.g. [4, 1]). Ontology-based keyword search of patient records has recently received considerable attention. Apart from our previous work [5] on the efficiency of ontology-based keyword searches over relational databases, complimentary work has considered this problem in the context of semi-structured (XML) data [3]. However, none of the above considers both the detection of outliers in the input keywords and the ontology-based concept generalization, both of which have direct effect on the quality of the retrieved results. The work of Pedro et. al [6] considers a federation of ontologies to retrieve medical data. Our own work can be used in the federation to retrieve data from individual ontologies.

References

[1] S. Agrawal, S. Chaudhuri, and G. Das. Dbxplorer: A system for keyword-based search over relational databases. In *ICDE*, pages 5–16, 2002.

[2] ClinicalTrials.gov. http://clinicaltrials.gov/.

[3] F. Farfán, V. Hristidis, A. Ranganathan, and R. P. Burke. Ontology-aware search on xml-based electronic medical records. In *ICDE*, pages 1525–1527, 2008.

[4] V. Hristidis and Y. Papakonstantinou. Discover: keyword search in relational databases. In *VLDB*, pages 670–681, 2002.

[5] A. Kementsietsidis, L. Lim, and M. Wang. Supporting ontology-based keyword search over medical databases. In *AMIA*, pages 409–413, 2008.

[6] V. Pedro, L. V. Lita, S. Niculescu, B. Rao, and J. G. Carbonell. Federated ontology search for the medical domain. In *OTM Workshops (1)*, pages 554–565, 2007.