Hierarchical Cost-sensitive Web Resource Acquisition for Record Matching

Yee Fan Tan
School of Computing
National University of Singapore
13 Computing Drive, Singapore 117417
tanyeefa@comp.nus.edu.sg

Min-Yen Kan
School of Computing
National University of Singapore
13 Computing Drive, Singapore 117417
kanmy@comp.nus.edu.sg

***Abstract*—Web information is increasingly used as evidence in solving various problems, including record matching. However, acquiring web-based resources is slow and can incur other access costs. As such, solutions often acquire only a subset of the resources to achieve a balance between acquisition cost and benefit. Unfortunately, existing work has largely ignored the issue of which resources to acquire. They also fail to emphasize on the hierarchical nature of resource acquisitions, *e.g.*, the search engine results for two queries must be obtained before their TF-IDF cosine similarity be computed. In this paper, we propose a framework for performing cost-sensitive acquisition of resources with hierarchical dependencies, and apply it to the web resource context. Our framework is versatile, and we show that a large variety of problems can be formulated using *resource dependency graphs*. We solve the resource acquisition problem by casting it as a combinatorial search problem. Finally, we demonstrate the effectiveness of our acquisition framework on record matching problems of different domains.**

I. Introduction

There are many real-life problems in which the given information is either insufficient or incomplete. Therefore, an increasing number of solutions elect to acquire additional information from external resources, such as by querying the web through a search engine, to achieve a better solution quality. Such works solve problems in vastly different application areas, and many of these solve problems (or subproblems) that can be seen as record matching. Examples include: the matching of entities to concepts in ontology creation [1], measuring semantic similarity between words [2], social network extraction [3], record linkage [4], author name disambiguation in bibliographic citations [5], and web people search [6]. All of these works demonstrated that acquiring additional information through a search engine resulted in increased matching effectiveness. However, acquiring such web information is time consuming due to slow web accesses and rate limiting by search engines, and can entail other access costs. This can cause significant bottlenecks unless web resources are acquired selectively, yet existing work has ignored this issue to a large extent.

This work was partially supported by a National Research Foundation grant "Interactive Media Search" (grant #R-252-000-325-279).

A way to limit the cost of resource acquisitions is by blocking, which filters out obvious mismatched record pairs before performing matching on the remainder (*e.g.*, [7]). Successful applications of blocking achieve a reasonable task performance with a greatly reduced number of pairwise matchings [8], which can reduce resource acquisitions significantly. However, blocking techniques are often ad-hoc and domain-specific, and can be difficult to define properly; poor blocking decisions can degrade performance [8].

On the other hand, there is also a large pool of work that formulates such resource acquisition problems into selective and cost-sensitive acquisition of missing attribute values in a classification model (*e.g.*, [9], [10]). While such works provide more principled acquisition algorithms, they generally assume that each attribute value in each instance is an independent resource. However, this is not true in the web resource context, *e.g.*, the hit count (number of web pages matching a query) for a query a can be a common attribute value for all pairwise instances that compare a with another item. Also, such works ignore possible hierarchical dependencies between resource acquisitions, *e.g.*, acquiring the hit count of a requires first acquiring the search engine results of a, and these search engine results can also be used to generate other attribute values such as similarity metrics between a and some other query.

Our contributions in this work are as follows: 1) We propose a framework for performing cost-sensitive resource acquisition problems with hierarchical dependences, through the use of a *resource dependency graph*. Our framework is versatile and can apply to a large variety of problems, and we show that a number of problems involving selective resource acquisitions can be formulated using resource dependency graphs. 2) Given a resource dependency graph of a record matching problem, and a benefit function that measures the goodness of making resource acquisitions, we propose an algorithm for making resource acquisitions that take the unique characteristics of such graphs into account. We solve the problem by applying the widely-studied Tabu search algorithm [11].

This paper is organized as follows. In Section II, we first review related work. Section III describes our cost-sensitive resource acquisition framework and its possible

applications. In Section IV, we propose an algorithm for solving the resource acquisition problem in the context of our framework. In Section V, we explain the benefit function we use for record matching problems. In Section VI, we evaluate the effectiveness of our acquisition framework, before concluding the paper.

II. Related Work

Record matching is widely studied and several papers have surveyed approaches to it (*e.g.*, [7], [12]). Here, we only focus on related work that utilizes web information to perform matching tasks. Again this idea is not new and hence we only survey a representative sample.

In [1], an ontology building system is created to match the entities extracted from an input document D to a list of concepts. For an entity $e \in D$ and a concept c, the system submits queries formed by e concatenated with patterns associated with c to a search engine. Concept c then gets a vote if the similarity of a retrieved snippet and D exceeds a threshold. Finally, e will be associated with the concept with the highest number of votes.

In [2], the semantic similarity between two words a and b is measured by first obtaining hit counts from the three queries a, b, and $a \wedge b$. Using these hit counts, web versions of similarity metrics can be computed. The values of these metrics then become attribute values in a test instance which is classified using a SVM classifier.

In [4], a framework for finding matching entities in an input list is given. A representative token t_c is selected from the input, and then the search engine is queried with the query $e \wedge t_c$ for each entity e in the list. Extensive experimentation is performed using various similarity metrics computed from hit counts, snippets, and web pages.

Through these sample works (and many others, *e.g.*, [13], [3], [14], [6]), a clear pattern can be seen. To perform record matching, queries of the form a, b, and $a \wedge b$ are issued to the search engine for some or all record pairs a and b in the input. Optionally, these queries can be further augmented with additional terms or tokens t, thus we may query with $a \wedge t$ instead of just querying a. Then similarity metrics or other information such as frequency counts may be extracted from the search engine results, which may be used standalone or combined to form test instances for a classifier such as a SVM.

However, obtaining search engine results and downloading web page are time consuming processes. Further, certain search engines perform rate limiting and restrict the number of queries one may make in one day. For example, Yahoo! Search has a daily quota of $5{,}000$ queries. However, even two lists of 100 items each can generate $10{,}000$ pairwise queries of the form $a \wedge b$, which requires two days. As such, it is necessary to acquire web resources in a selective manner. In our earlier work, we considered an adaptive querying framework [15] where an algorithm stronger in matching performance is combined with another algorithm faster in running time to produce a combined algorithm with good performance and with reasonable runtime. In the context of this paper, we can cast these two algorithms as producing two different attributes for test instances.

A related problem is the cost-sensitive acquisition of missing attribute values in instances (*e.g.*, [9], [10]), if we see missing attribute values as a kind of resource. However, in record matching problems, different instances often share common attribute values or have attribute values derived from some common resources, and such works do not consider this aspect. [16] solved a clustering problem expressed as a complete graph, where additional web resources can be acquired and added as additional vertices in the graph. However, they only considered one kind of additional resource (search engine results) and did not account for varying acquisition costs that differing resources may incur. Further, all of these cost-sensitive acquisition works do not consider hierarchical dependencies among resources, *e.g.*, web page downloads require acquiring search engine results first. To the best of our knowledge, our work in this paper is the first to generalize the resource acquisition problem to handle the conditions modeled in our framework.

III. Resource Acquisition Framework

In this section, we define our cost-sensitive framework, which is designed to handle many kinds of resource acquisition problems, including record matching problems involving web-based resources. Central to our framework is the notion of a *resource dependency graph*. We illustrate how resource dependency graphs can be constructed for a large variety of problems. We propose an algorithm for solving the acquisition problem in Section IV.

A. Our Framework

To make our description concrete, we use an instance of record linkage as a running example. Here, the input are two lists A and B, and the aim is to determine which record pairs $(a, b) \in A \times B$ are matches. As shown earlier, we may query a search engine with the queries a, b, or $a \wedge b$, and obtain raw search engine results which we denote as $search(a)$, $search(b)$, and $search(a \wedge b)$ respectively. We can then extract information from these search engine results, and construct a test instance $\mathbf{x}_{a,b}$ to be classified as a match or mismatch by a classifier. For concreteness, we assume that $\mathbf{x}_{a,b}$ has four attribute values: $hitcount(a)$, $hitcount(b)$, $hitcount(a \wedge b)$, and $dice(a, b)$. The first three are the hit counts returned by the search engine in $search(a)$, $search(b)$, and $search(a \wedge b)$ for the respective queries; and the fourth is the Dice coefficient $dice(a, b) = \frac{2 \cdot hitcount(a \wedge b)}{hitcount(a) + hitcount(b)}$.

In this example, we consider each of $search(a)$, $search(b)$, $search(a \wedge b)$, $hitcount(a)$, $hitcount(b)$, $hitcount(a \wedge b)$, and $dice(a, b)$ as a different type of resource

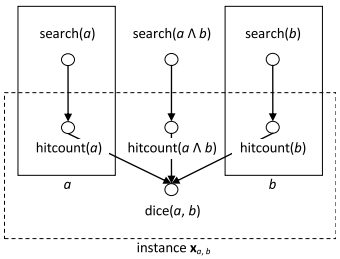

Figure 1. Structure of example resource dependency graph.

that may be acquired. The acquisition costs of different resources may not be uniform, *e.g.*, querying a search engine to obtain $search(a)$ takes significantly longer than extracting $hitcount(a)$ from $search(a)$ after it is acquired. Also, the classifier may classify an instance correctly with only a subset of its attribute values acquired. Further, some resource acquisitions have dependencies, *e.g.*, acquiring $hitcount(a)$ requires $search(a)$ to be acquired first. These dependencies can be captured in a resource dependency graph.

Definition 1. A *resource dependency graph* is a directed acyclic graph $G = (V, E)$, where each vertex in the vertex set V represents a resource, and an edge $u \rightarrow v$ exists in the edge set E if resource u must be acquired before resource v can be acquired. A subset V' of the vertex set V is called a *feasible vertex set* if for all vertices $v \in V'$, $u \rightarrow v$ implies that $u \in V'$. We denote the set of all feasible vertex sets in the graph G by $F(G)$, with $F(G) \subseteq 2^V$.

In other words, a vertex may be acquired only after all its ancestor vertices are acquired, and a feasible vertex set can be acquired as it is without violating any acquisition dependencies.

Resource dependency graphs for record matching problems exhibit some regularity in its structure, which allows us to present them in a compact manner using a variant of the plate notation regardless of the size of the input lists. The resource dependency graph of our running example is presented in Figure 1. Here, the dashed box indicates the vertices that form the attribute values of the test instance $\mathbf{x}_{a,b}$, and the solid box labeled a indicates that the vertices inside the box are common to all instances involving a, and likewise for the solid box labeled b. Hence, there are $|A|$ vertices of the form $search(a)$ as opposed to $|A| \times |B|$ vertices of the form $search(a \wedge b)$. Observe how some resources are common to multiple instances, *e.g.*, for each $a \in A$, $hitcount(a)$ is common to $|B|$ test instances.

Definition 2. A *resource acquisition problem* takes in the following as input:

- A resource dependency graph $G = (V, E)$.
- An *acquisition cost* function $cost : V \rightarrow \mathbb{R}^+ \cup \{0\}$. We assume that $cost(v) = 0$ for any acquired vertex v.
- A *benefit* function $benefit : F(G) \rightarrow \mathbb{R}$. We assume that $benefit(\emptyset) = 0$ and $benefit(V') = benefit(V'_u)$, where V'_u is the set of unacquired vertices in V'.
- A *vertex type* function $type : V \rightarrow T$, where T is a finite set of vertex types.

The aim of the resource acquisition problem is to acquire a feasible subset of vertices $V' \in F(G)$, with $cost(V') \leq budget$, such that the objective function $obj(V') = benefit(V') - cost(V')$ is maximized.

In our running example, there are six vertex types, corresponding to $search(a)$, $search(b)$, $search(a \wedge b)$, $hitcount(a)$, $hitcount(b)$, $hitcount(a \wedge b)$, and $dice(a, b)$. Depending on problem requirements, we can scale the acquisition costs and benefit values appropriately. Because search engine results take significantly longer to acquire compared to extracting hit counts, we may set the acquisition cost of a $search(\cdot)$ vertex to be greater than any other vertex. Also, as a classifier can only understand and process the attribute values in test instances, it cannot directly utilize the raw search engine results $search(a)$, $search(b)$, and $search(a \wedge b)$ since they are not part of the attribute values. Therefore, these vertices will have zero benefit value.

In most problems, we can assume that the resource dependency graph and the acquisition cost function are known and fixed. However, the benefit function is typically only a heuristic that approximates the true performance metric of the problem, as knowing the true metric entails knowing the solution. For example, in a record matching problem, we may want to minimize the total misclassification cost or to maximize the F-measure, but the benefit function may just return an estimated increase in classification confidence instead. In this paper, when we devise an algorithm for solving the resource acquisition problem, we treat the benefit function as a black box. Nonetheless, applications of our resource acquisition framework require a benefit function, and we explain the benefit function we use for record matching problems in Section V.

B. Applications

As resource acquisition problems can be modeled as resource dependency graphs, our framework can be applied to various scenarios. Our framework is not restricted to classification problems, as long as we have a benefit function that indicates how useful vertices are. We now suggest how resource dependency graphs can be constructed for record matching problems with other types of resources and attributes, as well as for a clustering variant.

Term frequencies obtained from web pages. Term frequencies (TF) and TF-IDF values from web pages are commonly used in record matching applications. Here, for a query q, we can add two types of resources: $webpage(q)$, representing the set of web pages that may be downloaded from the URLs in the top-k search engine results of $search(q)$; and $tf(q)$, representing the term frequencies of

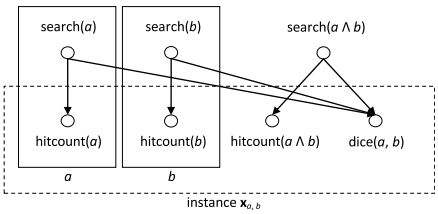

Figure 2. Alternative structure of example resource dependency graph.

the terms found in the downloaded web pages $webpage(q)$. Here, $webpage(q)$ depends on $search(q)$, and $tf(q)$ depends on $webpage(q)$. The term frequencies in $tf(\cdot)$ can then be used as attribute values in the test instances. If TF-IDF values are desired instead, then the $tf(q)$ vertices can be replaced by $tfidf(q)$ vertices, with the IDF values pre-calculated when the classification model is trained. Another alternative is to compute the cosine similarity between the term frequency or TF-IDF vectors; such $cos(a, b)$ vertices can be added to the resource dependency graph easily.

Common n-grams. The queries formed from a list of records may contain many common contiguous subsequences of tokens, or n-grams. Querying on these n-grams may return results for most or all of the original list queries, allowing us to make fewer queries compared to making the original queries individually [15]. As such, we may construct a vertex $search(g)$ for each frequently occurring n-gram g, with additional vertices $value(q, g)$ for information extracted from $search(g)$ pertaining to the original query q.

Clustering. Another form of record matching takes in a list L as input and determines which record pairs $(a, b) \in L \times L$ are matches. The resource dependency graph can be constructed similarly. However, instances in clustering problems are symmetric, *i.e.*, $\mathbf{x}_{a,b}$ and $\mathbf{x}_{b,a}$ are equivalent. Hence, $hitcount(a)$ and $hitcount(b)$ make poor attribute values since their positions are interchanged in $\mathbf{x}_{a,b}$ and $\mathbf{x}_{b,a}$, confusing classifiers such as linear SVMs. Instead, we can consider using the smaller and larger of $hitcount(a)$ and $hitcount(b)$ as attribute values.

C. Observations on Graph Structure

We make some observations on the structure of resource dependency graphs for record matching problems involving web resources.

We first observe that a resource dependency graph for web resource acquisitions typically has only 2 to 4 levels. Furthermore, these resource dependency graphs can be collapsed into 2 levels, where the root level mainly consists of vertices that are costly to acquire with zero benefit value (*e.g.*, $search(\cdot)$ and $webpage(\cdot)$); in contrast, the leaf level mainly consists of vertices that can be acquired very cheaply if their parent vertices have been acquired (*e.g.*, $hitcount(\cdot)$), and have positive benefit. For example, the running example of Figure 1 can be remodeled as Figure 2 with little difference. This is because all of $hitcount(a)$, $hitcount(b)$, $hitcount(a \wedge b)$, and $dice(a, b)$ can be acquired very cheaply given the search engine results, and thus the acquisition costs are dominated by $search(a)$, $search(b)$, and $search(a \wedge b)$. As another example, if the original graph contains the edges $search(a) \rightarrow webpage(a)$ and $webpage(a) \rightarrow tfidf(a)$, then we can restructure the edges as $search(a) \rightarrow tfidf(a)$ and $webpage(a) \rightarrow tfidf(a)$, without changing the optimal solution to the problem.

We also observe that the vertex out-degree may vary widely, leading to bushy topologies. To illustrate this observation we again consider the resource dependency graph in Figure 2. A vertex for $search(a \wedge b)$ typically only have very few child vertices, such as $hitcount(a \wedge b)$. On the other hand, a vertex for $search(a)$ can have at least $O(|B|)$ descendant vertices such as $dice(a, b)$. An algorithm that decides which vertices to acquire from the top down would need to figure out whether a vertex with many descendant vertices is worth acquiring over another vertex with fewer descendant vertices. This is because in a typical scenario where the descendant vertices can consist of several types, not all descendant vertices are equally good.

These characteristics make challenging resource acquisition problems. We found that simple algorithms such as acquiring vertices with the best cost-benefit ratio do not perform well. Therefore, we need a more sophisticated acquisition algorithm.

IV. Solving the Resource Acquisition Problem for Record Matching

The resource acquisition problem is essentially a combinatorial search problem over all subsets of V. However, a brute-force enumeration of all feasible subsets of V is prohibitively expensive even for a moderate-sized graph G. For example, in the simple resource acquisition graph of Figure 2, there are $2^{|A|} \times 2^{|B|} \times 4^{|A| \times |B|}$ possible acquisition decisions, even if we only consider the vertices that are attribute values in the test instances. Therefore, we need to employ some form of heuristic search.

In our heuristic search algorithm, we use $F(G)$, the set of feasible subsets of the vertex set V, as our state space. We denote the current state by V', and set $V' = \emptyset$ as our initial state. However, the unique structure of the resource dependency graph and the benefit function in record matching problems poses special difficulty as it is easy for the search algorithm to get stuck in the large number of local maxima, and keep revisiting the states around these maxima. For example, $V' = \emptyset$ is itself a local maxima, and from this state there is no guidance on which root vertices to add since typically all root vertices have zero benefit value. For another example, consider the resource dependency graph of Figure 2, and consider the state containing a $search(a)$ vertex. If one of the best vertices to acquire is $hitcount(b)$,

then it is easy for the algorithm to add $hitcount(a)$ into the current state and get stuck in a local maximum.

A solution must be able to address these topological issues in our web acquisition dependency graphs. We tackle these issues as follows: 1) We apply a widely-studied metaheuristic search algorithm known as Tabu search [11]. This search algorithm avoids revisiting the same states, allowing a better exploration of the state space. 2) We define a set of legal moves that are more intelligent. These moves allow the search algorithm to reach states that includes the leaf vertices faster. 3) We introduce a surrogate benefit function that propagates the benefit values from the leaf vertices upwards to the root vertices. This guides the search algorithm towards vertices with the highest benefit values.

A. Application of Tabu Search

Tabu search can be seen as a modification of simple hill climbing to encourage state space exploration, so as to avoid revisiting states and getting stuck in local maxima. It has been applied successfully in a number of combinatorial search problems (see [11] and the references therein). Like simple hill climbing, Tabu search uses the set of legal moves to generate neighbors of the current state, and then chooses the move that leads to the neighboring state with the highest objective value. However, Tabu search also maintains a list of prohibited moves called *Tabu list*. When a move is applied at some iteration, the move(s) that undo the effect of this move will be placed in the Tabu list for some number of iterations known as the *Tabu tenure*. As an exception, known as *aspiration criterion*, a move in the Tabu list is allowed if the resultant state has an objective value better than the visited states.

In our application of Tabu search, for better exploration of the state space, we allow Tabu search to enter a state whose total cost exceeds the budget, but penalize it heavily. We use a static Tabu tenure of 7 (used in many other Tabu search applications), and stop the search when 20 consecutive iterations have occurred without improvement in the best objective value. We then acquire the vertices in the state with the highest objective value.

B. Legal Moves

The set of feasible legal moves from a current state V' is exponential in size. Another key aspect in our design is to provide a relatively small number of potentially useful legal moves to reduce the search space in such large resource dependency graphs. For $V' \neq \emptyset$, let $D(V')$ be the set of children vertices of vertices in V' that are not in V'. For $V' = \emptyset$, we let $D(V')$ to be the set of root vertices in G instead. The legal moves in our framework are: 1) $Add(v)$: For a vertex $v \in D(V')$, add v and all its ancestors to V'. 2) $Remove(v)$: For a vertex $v \in V'$, remove v and all its descendants from V'. 3) $AddType(t)$: For a vertex type $t \in T$, construct a maximum subset of vertices $U \subseteq D(V')$ with vertex type t, such that the acquisition cost of V', U, and the ancestors of U does not exceed $budget$. Then add all the vertices in U and their ancestors to V'.

Note that all of these moves can batch add or remove multiple vertices in one operation. The aim for defining the $Add(\cdot)$ and $Remove(\cdot)$ this way is to allow the search algorithm to reach the vertices with high benefit values more easily. For example, in Figure 2, if the current state V' has a vertex $search(a)$, it might be worthwhile to add $dice(a, b)$ as well, but this move would not be possible if we define our legal moves to add only one vertex at a time unless V' already contains $search(b)$ and $search(a \wedge b)$ as well. Finally, $AddType(\cdot)$ allows as many descendant vertices of the same type to be added as possible, because it may be more beneficial to acquire one costly root vertex u and many cheap child vertices of u, rather than acquiring only u and one child vertex of u.

As for the Tabu moves, when a move adds a set of vertices V'' to the current state, we add any move that removes any vertex in V'' to the Tabu list; and when a move removes a set of vertices V'' from the current state, we add any move that adds any vertex in V'' to the Tabu list.

C. Surrogate Benefit Function

We now turn to the question on choosing a legal move from the current state V'. Many of the root vertices share the same acquisition cost and have zero benefit, giving no guidance on which root vertices lead to the best leaf vertices. Therefore, from the leaf vertices, we propagate benefit values upwards from vertices to edges and from edges to vertices, eventually reaching the root vertices. This allows us to define a surrogate benefit function and a surrogate objective function that we use to select the best legal move.

Definition 3. Let $0 \leq \lambda \leq 1$ be a *propagation factor* and $0 \leq \beta \leq 1$ be an *aggregation factor*. For a vertex $v \in V$, let $pa(v)$ and $ch(v)$ be the parent and child vertices of v respectively. We define the following recursively:

The *propagated benefit* of a vertex $v \in V$ is:

$$prop\text{-}benefit(v) = benefit(v) + max\text{-}pb(v) + \beta \cdot rest\text{-}pb(v)$$

where:

$$max\text{-}pb(v) = \max_{u \in ch(v)} prop\text{-}benefit(v \to u)$$

$$rest\text{-}pb(v) = \sum_{u \in ch(v)} prop\text{-}benefit(v \to u) - max\text{-}pb(v)$$

The *propagated benefit* of an edge $u \to v \in E$ is:

$$prop\text{-}benefit(u \to v) = \frac{\lambda \cdot (prop\text{-}benefit(v) - cost(v))}{|pa(v)|}$$

The propagation factor λ controls how much of the benefit value is propagated upwards, and the aggregation factor β controls the balance of aggregating from only the child with the maximum benefit value versus aggregating from

all children. Empirically, we determined that $\lambda = 0.5$ and $\beta = 0.1$ are effective, but we note that our search algorithm is not very sensitive to changes in these values.

Definition 4. Suppose the current state is $V' \in F(G)$. Let $V'' \subseteq V \setminus V'$ be a set of vertices such that $V' \cup V'' \in F(G)$ is a feasible subset. Let E' be the set of edges $v \rightarrow u$ with $v \in V''$ and $u \in V \setminus (V' \cup V'')$. Then the *surrogate benefit* function of adding V'' to the current state V' is:

$$surr\text{-}benefit(V', V'') = \sum_{v \rightarrow u \in E'} prop\text{-}benefit(v \rightarrow u)$$

Our surrogate benefit function $surr\text{-}benefit(V', V'')$ has two arguments instead of just a single V' argument. This allows the function to "forget" the propagated benefit values of V'' when the Tabu search moves to the new state $V' \cup V''$ when the actual benefit value of V'' is zero.

Definition 5. Suppose the current state is $V' \in F(G)$. The *surrogate objective* function of adding a set of vertices $V'' \subseteq V \setminus V'$ to the current state V', such that the new state $V' \cup V'' \in F(G)$ is also a feasible vertex set, is defined as $surr\text{-}obj(V', V'') = surr\text{-}benefit(V', V'') - cost(V'')$. The *surrogate objective* function of removing a set of vertices from the current state is defined similarly.

In the current state V', the Tabu search algorithm selects a legal move that adds or removes the vertices V'' with the maximum $surr\text{-}obj(V', V'')$ from the moves that are either non-Tabu or pass the aspiration criterion. When the search process terminates, we acquire the vertex set V' with the maximum objective value $obj(V', V'')$.

V. Benefit Function for Record Matching

To apply our resource acquisition framework to record matching problems, we need a benefit function. Here, a test instance $\mathbf{x}_{a,b}$ is constructed for each pair of records a and b. We assume that the classification of test instances are independent of each other, which importantly allows us compute the benefit function for a set of vertices V' from the expected decrease in misclassification costs for the corresponding test instances. For a test instance $\mathbf{x}$ and a subset A' of its missing attribute values, we use $E[mc(\mathbf{x})]$ and $E[mc(\mathbf{x} + A')]$ to denote the expected misclassification costs of $\mathbf{x}$ before and after acquiring A' respectively. Hence, $E[mc(\mathbf{x})] - E[mc(\mathbf{x} + A')]$ is the expected decrease in misclassification cost of $\mathbf{x}$ for acquiring A'.

Definition 6. Let G be a resource dependency graph and V' be a feasible subset of vertices in G. Let $I(V')$ be the set of test instances that contain one or more vertices in V' as attribute values, and let $A(V', \mathbf{x})$ denote the attribute values of the test instance $\mathbf{x}$ that are contained in V'. Then the *benefit* of V' is defined as:

$$benefit(V') = \sum_{\mathbf{x} \in I(V')} E[mc(\mathbf{x})] - E[mc(\mathbf{x} + A')]$$

In this paper, we used a linear SVM classifier because it is shown to be effective in many related works. To compute the quantities $E[mc(\mathbf{x})]$ and $E[mc(\mathbf{x}+A')]$ for the SVM classifier, we applied the method given in [17]. The advantage of the method in [17] over other cost-sensitive acquisition methods such as [9] and [10] is that it can compute $E[mc(\mathbf{x} + A')]$ efficiently for an arbitrary subset A' of missing attribute values in any test instance $\mathbf{x}$.

In our evaluation, we used this benefit function to evaluate our resource acquisition algorithm.

VI. Evaluation

We evaluated our algorithm on three datasets in two different domains. The first two datasets, SL-GENOMES[1] and SL-DBLP[2], are record linkage problems whose input lists SF and LF are short forms and long forms respectively. The aim is to match short forms $sf \in SF$ (*e.g.*, "WI-IAT") to their long forms $lf \in LF$ (*e.g.*, "International Conference on Web Intelligence and International Conference on Intelligent Agent Technology"). The size of SL-GENOMES is 307×307 while the size of SL-DBLP is 906×920. The third dataset, AUTHOR-DBLP, is an author name disambiguation task whose input are $L_1, \ldots, L_n$, where each L_i is a list of publication records containing the author name s_i which may represent one or more individuals. The clustering-style task here is to determine which pairs of publication records $(a, b) \in L_i$ belongs to the same individual. This dataset is created by randomly selecting 352 author names from DBLP[3]. These are important problems with real-world applications, and where useful information is scattered all over the web.

The characteristics and resource dependency graphs of these datasets are shown in Table I and Figure 3 respectively. We use Google SOAP Search API and retrieved the top-10 results for each query. For SL-GENOMES and SL-DBLP, each query is made up of a short form or a long form plus a few domain specific keywords to promote results of that domain. For AUTHOR-DBLP, each query is the title of a publication record. Each dataset is split into equal training and testing halves; the training half is used to train the SVM and the resource dependency graph is built on the testing half and evaluated. During training, five-fold cross-validation is performed to find the optimal SVM regularization parameter. Some of vertex labels in Figure 3 are self-explanatory; others need a bit of explanation: *e.g.*, $minhitcount(a, b)$ and $maxhitcount(a, b)$ are the minimum and maximum of $hitcount(a)$ and $hitcount(b)$ respectively; $cosine(a, b)$ is $\frac{hitcount(a \wedge b)}{\sqrt{hitcount(a) hitcount(b)}}$; $snippetcount(a \rightarrow b)$ and $snippetcount(a \leftarrow b)$ are the number of snippets in $search(a)$ and $search(b)$ containing b and a respectively;

[1] http://www.ornl.gov/sci/techresources/Human_Genome/acronym.shtml
[2] http://dblp.uni-trier.de/db/conf/indexa.html
[3] http://dblp.uni-trier.de/xml/

Table I
THE EVALUATION DATASETS.

Dataset	SL-GENOMES	SL-DBLP	AUTHOR-DBLP
Test instances	23,409	213,444	27,046
Matching pairs	307	926	20,582
Vertices (unacquired)	117,657	1,069,068	275,220
Edges	140,760	1,281,588	595,012
Misclassification cost	50 / 500	50 / 1,500	1,000 / 200

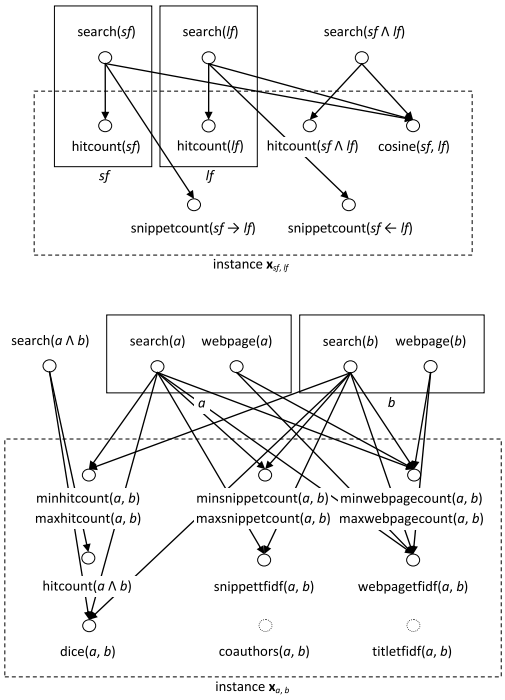

Figure 3. Structure of resource dependency graphs for SL-GENOMES (top), SL-DBLP (top), and AUTHOR-DBLP (bottom). Dotted vertices indicate pre-acquired vertices before starting the acquisition algorithm.

$snippettfidf(a, b)$ is the TF-IDF cosine similarity between the tokens of the snippets in $search(a)$ and $search(b)$; and $coauthors(a, b)$ is the number of common coauthors between two publication records a and b. Other vertices are defined in an analogous manner. For AUTHOR-DBLP, all $coauthor(\cdot, \cdot)$ and $titletfidf(\cdot, \cdot)$ vertices were pre-acquired prior to executing the acquisition algorithm. Note that SL-DBLP has over 1 million vertices and 1 million edges. For our experiments, we set the acquisition costs of each $search(\cdot)$, $webpage(\cdot)$, and other unacquired vertex to 10, 100, and 1 respectively.

We evaluated our algorithm against five baselines:

- **Random.** This algorithm acquires vertices at random until the budget is reached.
- **Least cost.** This algorithm acquires the least cost vertices until the budget is reached.
- **Best benefit.** This algorithm acquires the vertices with the best benefit until the budget is reached.
- **Best cost-benefit ratio.** This algorithm acquires the vertices with the best benefit to cost ratio until the budget is reached.
- **Best type.** This algorithm acquires the maximum number of vertices of one type not exceeding the budget that maximizes the objective value.

We evaluated each algorithm by starting with a resource dependency graph with no vertices acquired, executing the algorithm for 200 iterations, each time with a budget of 100 for SL-GENOMES and SL-DBLP and 1,000 for AUTHOR-DBLP. Each execution generates a data point, consisting of the cumulative acquisition cost of all the vertices acquired so far and the misclassification cost of all the instances. All data points for each algorithm are then plotted in a chart of total misclassification cost against total acquisition cost, enabling us to compare the performance between the different algorithms. We also include in these charts a line labeled **Manual**, which estimates the best possible performance for a manual process, given the knowledge of which attribute value acquisitions will allow which test instances to be correctly classified by the SVM classifier. This manual process yields one data point, which we join with the point where no resource acquisitions have taken place. Naturally, this manual process outperforms any automated algorithm.

The results are shown in Figure 4. It is desirable for an algorithm to incur less misclassification cost using less acquisition cost. These results show that our algorithm consistently and significantly outperforms all the baseline algorithms evaluated against in all the three datasets. From these charts, we also note that the baselines perform quite differently, which gives strong evidence that the three datasets have quite different characteristics. Each dataset has some resources that are more useful than others, and different datasets favour the acquisition of different kinds of resources. For example, the least cost algorithm is the second-best in SL-GENOMES and much of SL-DBLP, but is tied with best benefit as the worst algorithm in AUTHOR-DBLP. On the other hand, while the best type algorithm is the second-best in AUTHOR-DBLP and a small part of SL-DBLP, it is the second-worst algorithm in SL-GENOMES. Despite the differences in dataset characteristics, our algorithm incurs the least misclassification cost over the entire range of acquisition costs plotted in all three charts.

Next, we consider how much our algorithm improves over the second-best algorithm, by considering the difference in misclassification cost between the second-best algorithm and the manual process and the difference in misclassification cost between our algorithm and the manual process. We compute the average improvement in the difference when our algorithm is used instead of the second-best. This average is computed using interpolated misclassification costs for different acquisition costs, for acquisition costs in intervals of 1,000. Our algorithm makes an average improvement of more than 49%, 29%, and 74% over the second-best algorithm for SL-GENOMES, SL-DBLP, and AUTHOR-DBLP

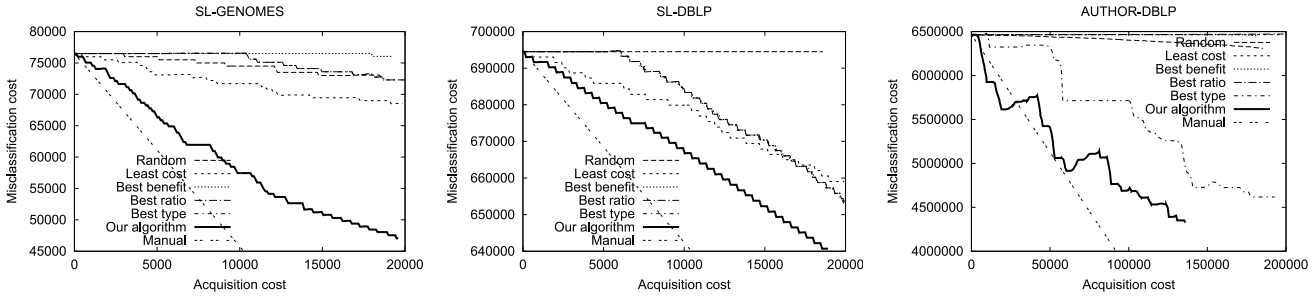

Figure 4. Experimental results for SL-GENOMES (left), SL-DBLP (middle), and AUTHOR-DBLP (right).

respectively. The significant improvements demonstrate the effectiveness of our algorithm over the baselines.

We now examine the search engine queries (*i.e.*, acquisition of $search(\cdot)$ vertices) issued by our algorithm. For this, we consider the test instances that are correctly classified when our acquisition algorithm is used, and compare it with acquiring all the attribute values of these instances. For all three datasets, we found that our algorithm acquires less than 10% of the $search(\cdot)$ vertices, saving more than 90% of the search engine queries. This is because each $search(a)$ or $search(b)$ vertex services a large number of test instances, whereas each $search(a \wedge b)$ vertex services only a single test instance. As most of the instances do not require the information in $search(a \wedge b)$ to be classified correctly, our algorithm mostly made search engine calls of the form $search(a)$ or $search(b)$, resulting in significantly less queries being made. This is important not only for saving time, but also allows more work to be done with the search engine in the face of daily quotas or rate limiting.

VII. Conclusion

In this paper, we introduced a hierarchical cost-sensitive resource acquisition framework, which models dependencies in resource acquisitions through a resource acquisition graph. Our resource acquisition framework is versatile, applicable to many different problems, as long as a benefit function can be supplied. We showed that resource acquisition graphs for record matching problems involving web resources have a challenging structure. This requires us to devise a search algorithm that can answer these challenges. Our evaluation on a short form to long form matching problem and an author name disambiguation problem shows that our acquisition algorithm significantly outperforms a number of baseline algorithms.

References

[1] P. Cimiano, G. Ladwig, and S. Staab, "Gimme' the context: Context-driven automatic semantic annotation with C-PANKOW," in *WWW*, 2005.

[2] D. Bollegala, Y. Matsuo, and M. Ishizuka, "Measuring semantic similarity between words using web search engines," in *WWW*, 2007.

[3] Y. Matsuo, J. Mori, M. Hamasaki, K. Ishida, T. Nishimura, H. Takeda, K. Hasida, and M. Ishizuka, "POLYPHONET: An advanced social network extraction system from the web," in *WWW*, 2006.

[4] E. Elmacioglu, M.-Y. Kan, D. Lee, and Y. Zhang, "Web based linkage," in *WIDM*, 2007.

[5] Y. F. Tan, M.-Y. Kan, and D. Lee, "Search engine driven author disambiguation," in *JCDL*, 2006.

[6] D. V. Kalashnikov, R. Nuray-Turan, and S. Mehrotra, "Towards breaking the quality curse: a web-querying approach to web people search," in *SIGIR*, 2008.

[7] W. E. Winkler, "Overview of record linkage and current research directions," U.S. Bureau of the Census, Tech. Rep., 2006.

[8] K. Goiser and P. Christen, "Towards automated record linkage," in *AusDM*, 2006.

[9] C. X. Ling, V. S. Sheng, and Q. Yang, "Test strategies for cost-sensitive decision trees," *TKDE*, vol. 18, no. 8, 2006.

[10] M. Saar-Tsechansky, P. Melville, and F. Provost, "Active feature-value acquisition," *Management Science*, vol. 55, no. 4, 2009.

[11] F. Glover, "Tabu search: A tutorial," *Interfaces*, vol. 20, no. 4, 1990.

[12] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios, "Duplicate record detection: A survey," *TKDE*, vol. 19, no. 1, 2007.

[13] M. Sahami and T. D. Heilman, "A web-based kernel function for measuring the similarity of short text snippets," in *WWW*, 2006.

[14] J.-H. Oh and H. Isahara, "Hypothesis selection in machine transliteration: A web mining approach," in *IJCNLP*, 2008.

[15] Y. F. Tan, E. Elmacioglu, M.-Y. Kan, and D. Lee, "Efficient web-based linkage of short to long forms," in *WebDB*, 2008.

[16] P. Kanani and A. McCallum, "Resource-bounded information gathering for correlation clustering," in *COLT*, 2007.

[17] Y. F. Tan and M.-Y. Kan, "Cost-sensitive attribute value acquisition for support vector machines," National University of Singapore, Tech. Rep., 2010.