A Framework for Hierarchical Cost-sensitive Web Resource Acquisition*

Yee Fan Tan and **Min-Yen Kan**
School of Computing
National University of Singapore
13 Computing Drive, Singapore 117417
{tanyeefa, kanmy}@comp.nus.edu.sg

March 2010

Abstract

Many record matching problems involve information that is insufficient or incomplete, and thus solutions that classify which pairs of records are matches often involve acquiring additional information at some cost. For example, web resources impose extra query or download time. As the amount of resources that can be acquired is large, solutions invariably acquire only a subset of the resources to achieve a balance between acquisition cost and benefit. At the same time, resources often have hierarchical dependencies between themselves, *e.g.*, the search engine results for two queries must be obtained before the TF-IDF cosine similarity between their snippets can be computed.

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, applicable to a large variety of problems. We show that many problems involving selective resource acquisitions can be formulated using *resource dependency graphs*. We then solve the resource acquisition problem by casting it as a combinatorial search problem. As the support vector machine is commonly used to effectively solve record matching problems, we also propose a benefit function that works with this classifier. Finally, we demonstrate the effectiveness of our acquisition framework on record matching problems.

1 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 [Cimiano et al., 2005], measuring semantic similarity between words [Bollegala et al., 2007], social network extraction [Matsuo et al., 2006], record linkage [Elmacioglu et al., 2007], author name disambiguation in bibliographic citations [Tan et al., 2006], and web people search [Kalashnikov et al., 2008]. 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.

One common approach to limit the cost of resource acquisitions is through *blocking*, which filters out obvious mismatches and only performs matching on non-obvious record pairs (*e.g.*, [Winkler, 2006]). Successful

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

applications of blocking can achieve a reasonable task performance with a greatly reduced number of pairwise matchings [Goiser and Christen, 2006]. Thus, the amount of required resource acquisitions can be reduced significantly. However, blocking techniques are often ad-hoc domain-specific techniques and can be difficult to define properly; poor blocking decisions can result in degraded performance [Goiser and Christen, 2006].

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.*, [Ling et al., 2006], [Saar-Tsechansky et al., 2009]). While such works provide more principled acquisition algorithms, they generally do not consider dependencies in resource acquisitions, and assume that resource acquisitions are independent events across different instances. However, such assumptions are often not true in the web resource context where resource acquisitions can have hierarchical dependencies. For example, search engine results for a and b must be acquired to compute web-based similarity metrics between a and b, but the acquired search engine results for a is also available for all pairwise instances that compare a with another item.

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 be applied 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 cast the problem as a combinatorial search problem, and we apply the widely-studied Tabu search algorithm [Glover, 1990] to solve it.

3. We propose a benefit function for applying our resource acquisition framework to record matching problems when the classification model is the support vector machine. We show that the our resource acquisition algorithm is effective in this setting.

This paper is organized as follows. In Section 2, we first review related work. In Section 3, we describe our cost-sensitive resource acquisition framework and its possible applications. In Section 4, we propose an algorithm for solving the resource acquisition problem in the context of our framework. In Section 5, we describe a support vector machine based benefit function for use in our framework. In Section 6, we evaluate the effectiveness of our acquisition framework. In Section 7, we conclude the paper.

2 Related Work

Record matching is widely studied and several papers have surveyed approaches to it (*e.g.*, [Winkler, 2006], [Elmagarmid et al., 2007], [Smalheiser and Torvik, 2009]). Here, we will only focus on related work that utilizes web information to perform matching tasks. Again this idea is not new and hence we will only survey a representative sample.

In [Cimiano et al., 2005], 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. From the retrieved snippets, if the similarity of a snippet and D exceeds a threshold then concept c gets a vote. Thus, e will be associated with the concept with the highest number of votes.

In [Bollegala et al., 2007], 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 support vector machine classifier.

In [Elmacioglu et al., 2007], a framework for finding matching entities in an input list is given. First, a representative token t_c is selected from the input data, 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.*, [Sahami and Heilman, 2006], [Matsuo et al., 2006], [Oh and Isahara, 2008], [Kalashnikov et al., 2008]), 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 query with 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 support vector machine.

However, obtaining search engine results and downloading web page are time consuming processes. Further, search engines typically 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 [Tan et al., 2008] 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, to be used by a classifier to determine whether these instances represent matches.

Thus, another line of related work is in the cost-sensitive acquisition of missing attribute values in instances, such as [Ling et al., 2006; Saar-Tsechansky et al., 2009]. However, none of these works considered the possibility of different instances sharing the same attribute values, or attribute values in different instances being derived from the same resource, as commonly found in the structure of record matching problems. [Kanani and McCallum, 2007] solved a clustering problem expressed as a complete graph, and where additional web resources can be acquired and added as additional vertices in the graph. However, their work only considered one kind of additional resource (search engine results) and did not account for varying acquisition costs that differing resources may incur. Further, another common issue in all of these cost-sensitive acquisition papers is that none of them considered the hierarchical dependency of resources, such as the fact that web page downloads depend on search engine results. To the best of our knowledge, our work is the first to generalize the resource acquisition problem to handle the conditions modeled in our framework.

3 Resource Acquisition Framework

Our cost-sensitive resource acquisition framework is designed to handle many kinds of resource acquisition problems. Central to our framework is the notion of a *resource dependency graph*. To aid our description, we use a simple record matching example involving web-based resources as our running example. We then expand on this example to illustrate how the resource dependency graphs can be constructed for other record matching problems. We define our framework in this section, and propose an algorithm for solving the acquisition problem in Section 4.

3.1 Our Framework

We use a particular kind of record matching problem known as record linkage as our running example. The input is two lists A and B, and the aim is to determine which record pairs $(a, b) \in A \times B$ are matches. As illustrated 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 (the number of web pages matching a query) 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 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

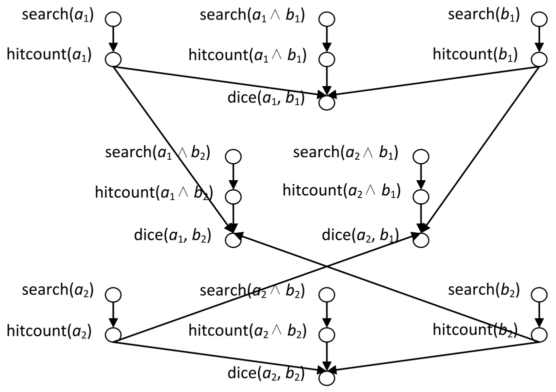


Figure 1: Example resource dependency graph for two lists of 2 records each.

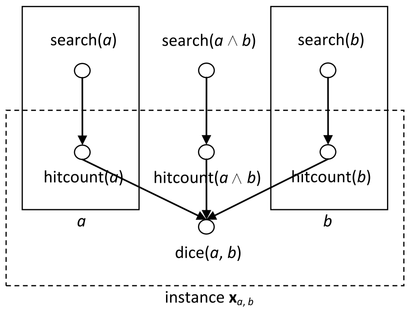


Figure 2: Structure of example resource dependency graph in plate notation.

longer than extracting $hitcount(a)$ from $search(a)$ after $search(a)$ is acquired. Also, we may not want to acquire all resources, as the classifier may already classify an instance correctly with a subset of the attribute values in an instance. 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 $u \to v$ exists in the edge set E if resource u must be acquired before v can be acquired. In other words, a vertex may be acquired only after all its ancestor vertices have been acquired. $\square$

For our running example, Figure 1 shows the resource dependency graph corresponding to two lists of 2 records each. Observe how some resources are common to multiple instances, *e.g.*, $hitcount(a_1)$ is common to the instances $\mathbf{x}_{a_1,b_1}$ and $\mathbf{x}_{a_1,b_2}$. We also see that in this simple case, the resource dependency graph already starts to get complex. However, resource dependency graphs for record matching problems exhibit structural regularity. Hence, we can use a variant form of plate notation and represent the resource dependency graph for input lists of any size using Figure 2. Here, the solid box labeled a means that the vertices inside the box are common to all instances involving a, and likewise for the box labeled b. Hence, there are $|A|$ vertices of the form $search(a)$ as opposed to $|A| \times |B|$ vertices of the form $search(a \wedge b)$. The dashed box indicates the vertices that form the attribute values of the test instance $\mathbf{x}_{a,b}$.

Before defining the resource acquisition problem proper, we first define some of its elements.

Definition 2. Given a *resource dependency graph* $G = (V, E)$, a set of vertices $V' \subseteq V$ is called a *feasible* vertex set if for all vertices $v \in V'$, we have $u \to v$ implies that $u \in V'$. That is, a feasible vertex set can be acquired without violating any acquisition dependencies. We denote the set of all feasible vertex sets in the graph G by $F(G)$, with $F(G) \subseteq 2^V$. $\square$

Definition 3. An *acquisition cost* function is a function $cost : V \to \mathbb{R}^+ \cup \{0\}$. We assume that $cost(v) = 0$ for any acquired vertex v. We also define a shorthand notation: for any feasible set of vertices $V' \subseteq F(G)$, $cost(V') = \sum_{v \in V'} cost(v)$. □

Definition 4. A *benefit* function is a function $benefit : F(G) \to \mathbb{R}$. We assume that $benefit(\emptyset) = 0$ and $benefit(V') = benefit(V'_u)$, where V'_u is the set of unacquired vertices in V'. We also define a shorthand notation: for any vertex v, $benefit(v) = benefit(\{v\})$. □

Definition 5. A *vertex type* function is a function $type : V \to T$, where T is a finite set of vertex types. □

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. We may (empirically or otherwise) set the acquisition costs of each $search(\cdot)$ vertex and each other vertex to 10 and 1 respectively, as search engine results take significantly longer to obtain compared to extracting hit counts. 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. Thus in a resource acquisition problem, we would like to acquire a subset of vertices that maximizes the difference between the benefit of acquiring these vertices and the total cost of acquiring them.

Definition 6. A *resource acquisition problem* takes in as input a resource dependency graph $G = (V, E)$, an acquisition cost function $cost(\cdot)$, a benefit function $benefit(\cdot)$, a vertex type function $type(\cdot)$, and a budget $budget$. The aim of this problem is to acquire a feasible subset of vertices $V' \in F(G)$, with $cost(V') \leq budget$, such that objective function $obj(V') = benefit(V') - cost(V')$ is maximized. □

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, because knowing the true metric function entails knowing the solution set. 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 consider algorithms for solving the resource acquisition problem, we treat the benefit function as a black box. However, good benefit functions should be (almost) monotonic, *i.e.*, $benefit(V_1 \cup V_2) \geq benefit(V_1)$ for all (or most) V_1 and V_2 that are subsets of the vertex set V, facilitating algorithms that acquires vertices in an incremental manner. In Section 5, we define a benefit function in the context of classification problems where missing attribute values in test instances may be acquired.

By modeling resource acquisition problems as resource dependency graphs, we have defined our framework so it can be applied to various problem and solution settings. Our framework does not insist on having a classifier to classify test instances, as long as we have a benefit function that indicates how useful vertices are. In the following, we suggest how resource dependency graphs can be constructed for record matching problems when additional types of resources and attributes are added, and when a variant of the record linkage problem is encountered.

Term frequencies obtained from web pages. Term frequencies 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 the terms found in the downloaded web pages $webpage(q)$. Of course, $webpage(q)$ 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 the TF-IDF values are desired instead, then the $tf(q)$ vertices can be replaced by $tfidf(q)$ vertices, with the document frequencies pre-calculated during a precomputation phase when the classification model is trained. Yet another alternative is to compute the cosine similarity between the term frequency or TF-IDF vectors; such $cos(a, b)$ vertices can be added into the resource dependency graph easily.

Common n-grams. It is possible that the queries formed from a list of records contain many common contiguous subsequences of tokens, or n-grams. If a particular n-gram is shared by at least k of the queries,

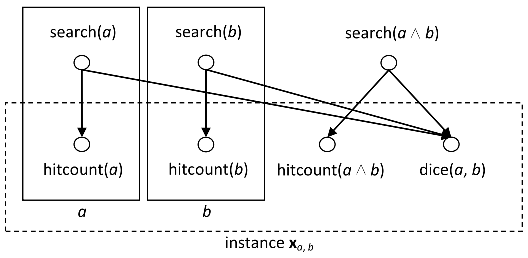


Figure 3: Alternative structure of example resource dependency graph.

then querying this n-gram may produce results for all these queries, allowing us to save on the number of queries compared to querying the original queries individually [Tan et al., 2008]. As such, we may construct a vertex $search(g)$ for each n-gram g that occurs in at least k of the original queries, with additional vertices of the form $value(q, g)$ for information extracted from $search(g)$ pertaining to the original query q.

Clustering. Another common 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 building of the resource dependency graph is analogous to that of record linkage. However, instances in clustering problems are symmetric, *i.e.*, $\mathbf{x}_{a,b}$ and $\mathbf{x}_{b,a}$ are equivalent. Hence, when constructing an instance $\mathbf{x}_{a,b}$, $hitcount(a)$ and $hitcount(b)$ make poor attribute values because the equivalent instance $\mathbf{x}_{b,a}$ would have the positions of these two attribute values interchanged, confusing classifiers such as linear support vector machines. Instead, we can create two new vertices for the smaller and larger of $hitcount(a)$ and $hitcount(b)$, and use them as attribute values instead.

3.2 Observations on Graph Structure

We first make some observations on the structure of resource dependency graphs involving web resources. These observations inform our proposed acquisition algorithm.

We first observe that a resource dependency graph typically has only 2 to 4 levels. Further, the vertices at or near the level of root vertices are often the results of costly web queries but with zero benefit value (*e.g.*, $search(\cdot)$ and $webpage(\cdot)$), while the vertices at or near the level of leaf vertices are typically attribute values that can be extracted from the raw information at negligible costs (*e.g.*, $hitcount(\cdot)$). As such, many resource dependency graphs can be restructured as hierarchies with 2 levels, where the root level mainly consists of vertices that are costly to acquire and the leaf level mainly consists of vertices that can be acquired very cheaply. For example, if the original graph contains the edges $search(a) \to webpage(a)$ and $webpage(a) \to tfidf(a)$, then we can restructure the edges as $search(a) \to tfidf(a)$ and $webpage(a) \to tfidf(a)$. This does not change the problem at all because acquiring $tfidf(a)$ still requires acquiring both $search(a)$ and $webpage(a)$, and merely acquiring $webpage(a)$ without $tfidf(a)$ is useless and produces a suboptimal solution. As another example, consider the original running example of Figure 2. 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, we can collapse these vertices yielding a two-layer graph as shown in Figure 3 that approximates the original with little difference, since the acquisition costs are dominated by $search(a)$, $search(b)$, and $search(a \wedge b)$.

The next observation is 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 3. 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 $dice(a, b)$.

4 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 3, 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. It is completely infeasible to do an exhaustive search. Therefore, we need to employ some form of heuristic.

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 local maxima. As an example consider the resource dependency graph of Figure 3, where only the leaf vertices have positive benefit value and each of them has benefit greater than cost. Suppose we start from an empty state $V' = \emptyset$, and only consider adding only a single vertex to the current state. Then only $search(\cdot)$ vertices can be added, which only incurs cost but no benefit. Further, there is no guidance on which $search(\cdot)$ vertex to add. Suppose $search(a)$ (for some $a \in A$) was added to the current state and one of the best vertex to acquire is $hitcount(b)$ (for some $b \in B$). Then it is easy for the algorithm to add $hitcount(a)$ into the current state and get stuck in another local maxima. With a large number of local maxima, it is easy for search algorithms to keep revisiting the states around the local maxima.

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 [Glover, 1990]. 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.

4.1 Application of Tabu Search

We first describe our application of Tabu search, which can be seen as a modification of simple hill climbing that avoids revisiting states and to escape local maxima. It has been applied successfully in a number of combinatorial search problems [Glover, 1990]. 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 implements a Tabu list, which stores the moves that are prohibited, or marked as Tabu. More specifically, when a particular move is applied at some iteration, the move(s) that undo the effect of the applied move will be placed in the Tabu list for a set number of iterations known as the Tabu tenure. This prevents the search process to repeatedly visit the same states and encourages exploration of the state space. However, as an exception, Tabu search allows a Tabu move to be applied to the current state if the resultant state has an objective value that is better than the states that have been visited so far. This *aspiration criterion* is applied because the resultant state is obviously an unexplored state. We stop the Tabu search when a certain number of consecutive iterations have occurred without improvement in the best objective value. We then acquire the vertices in the state with the highest objective value.

4.2 Legal Moves

The set of feasible legal moves from current state V' is often very large – 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 also not in V'. For example, in Figure 3, $D(\{search(a)\})$ contain the $hitcount(a)$ vertex and all $dice(a, b)$ vertices. 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 $Add(v)$ and $Remove(v)$ this way is to allow the search algorithm to reach the vertices with high benefit values more easily. For example, in Figure 3, 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 the current state already contains $search(b)$ and $search(a \wedge b)$ as well. Finally, $AddType(t)$ allows as many descendant vertices of the same type to be added as possible, because there are cases where the true benefit comes from acquiring one costly root vertex and many cheap leaf vertices rather than one costly vertex and only one cheap leaf vertex. As for the Tabu moves, whenever 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 whenever 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.

4.3 Surrogate Benefit Function

Having defined the set of legal moves, we now turn to the question of how to choose a 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, we propagate benefit values upwards from the leaf vertices to the root vertices, by repeatedly propagating from vertices to edges and then from edges to vertices. This allows us to define a surrogate benefit function $surr\text{-}benefit(V', V'')$ for the case when a subset of vertices V'' is added to the current state V' that takes into account the benefit values propagated from the descendant vertices of V''. Using this surrogate benefit function, we can then define a surrogate objective function $surr\text{-}obj(V', V'')$ that we use to select the best legal move to reach the next state.

Definition 7. 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

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

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

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

□

For a leaf vertex v, we have $prop\text{-}benefit(v) = benefit(v)$. This allows us to precompute all the propagated utilities as an efficient preprocessing step, using dynamic programming, computing from the leaves towards the root. 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 maximum benefit value versus aggregating from all children.

Definition 8. Suppose the current state is $V' \in F(G)$. Let $V'' \subset V \backslash V'$ be a set of vertices such that $V' \cup V'' \in F(G)$ is a feasible subset. Let $E'(V', V'')$ be the set of edges $v \to u$ 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:

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

□

Our surrogate benefit function $surr\text{-}benefit(V', V'')$ has two arguments V' and V'' 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 9. Suppose the current state is $V' \in F(G)$.

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

Thus, 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. To encourage exploration of the state space, we allow the Tabu search to enter a state V' whose total cost exceeds the budget, but we subtract from the surrogate objective value and objective value a large penalty proportional to $budget - cost(V')$. When the search process terminates, we acquire the vertex set V' with the maximum objective value $obj(V', V'')$.

5 A Support Vector Machine based Benefit Function for Record Matching

The final piece of the puzzle is to construct the benefit function for record matching problems where test instances $\mathbf{x}_{a,b}$ are constructed for the record pairs $(a, b) \in A \times B$. We operationalize this in the context of the support vector machine as it has been shown to be effective in many related works. While there is a significant pool of work in cost-sensitive selective acquisition, many of these use decision trees (*e.g.*, [Ling et al., 2006; Davis et al., 2006]), which considers acquiring only a *single* attribute value per iteration and dictates the order in which values are acquired. The prior work provides no guidance on estimating the reduction in misclassification cost for acquiring an *arbitrary* subset of attribute values in a test instance. This motivates our support vector machine based benefit function, and allow us to apply our work in cost-sensitive acquisition of missing attribute values in support vector machines [Tan and Kan, 2010]. Specifically, [Tan and Kan, 2010] gives a way to estimate the misclassification cost for a test instance $\mathbf{x}$, as well as the same instance with an (arbitrary) subset of its missing attribute values A' acquired, denoted by $E[mc(\mathbf{x})]$ and $E[mc(\mathbf{x} + A')]$ respectively.

We give a simplified description of $E[mc(\mathbf{x})]$ and $E[mc(\mathbf{x} + A')]$ for a support vector machine using the linear kernel here, and refer the reader to [Tan and Kan, 2010] for more details. For a support vector machine using the linear kernel, its decision function is:

$$f(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle + b$$

A trained support vector machine can be extended to provide the posterior probability that a test instance $\mathbf{x}$ belongs to the positive class or negative class, denoted by $p_+(\mathbf{x})$ or $p_-(\mathbf{x})$ respectively, with $p_+(\mathbf{x}) + p_-(\mathbf{x}) = 1$ [Platt, 2000]. We assume that prior to classification, each unacquired missing attribute value x_k in a test instance $\mathbf{x}$ is replaced by x'_k, the mean value of the attribute in the training data. For a test instance $\mathbf{x}$ with missing attribute values, we can define its *modified weight vector* $\mathbf{w}'(\mathbf{x})$, which is the weight vector $\mathbf{w}$ with the weights corresponding to the missing attribute values of $\mathbf{x}$ zeroed out. Let mc_+ and mc_- be the misclassification cost for misclassifying a positive and negative instance, respectively. The *expected misclassification cost* of a test instance $\mathbf{x}$ is defined as:

$$E[mc(\mathbf{x})] = \begin{cases} \left(1 - \frac{\|\mathbf{w}'(\mathbf{x})\|_2}{\|\mathbf{w}\|_2}) \cdot p_+(\mathbf{x})\right) \cdot mc_- & \text{if } f(\mathbf{x}) \geq 0 \\ \left(1 - \frac{\|\mathbf{w}'(\mathbf{x})\|_2}{\|\mathbf{w}\|_2}) \cdot p_-(\mathbf{x})\right) \cdot mc_+ & \text{otherwise} \end{cases}$$

The expected misclassification cost for $\mathbf{x}$ with a subset A' of its missing attribute values acquired, $E[mc(\mathbf{x} + A')]$, can be defined in a similar manner. In this paper, we use only the linear kernel, but we note that the expected misclassification cost of an instance can also be computed for support vector machines using an arbitrary nonlinear kernel [Tan and Kan, 2010].

Given $E[mc(\mathbf{x})]$ and $E[mc(\mathbf{x}+A')]$, we can then define our support vector machine based benefit function. We follow the approach of [Ling et al., 2006] and define the benefit of A' to be the expected decrease in misclassification cost of $\mathbf{x}$ for acquiring A'.

Definition 10. Let $\mathbf{x}$ be a test instance and let A' be a subset of its missing attribute values. The *benefit* of acquiring A' is defined as:

$$benefit(\mathbf{x}, A') = E[mc(\mathbf{x})] - E[mc(\mathbf{x} + A')]$$

$\square$

Here, we assume that the classification of test instances are independent of each other. This assumption importantly allows us to decompose the benefit function for a set of vertices V' into separate benefit functions for V' with respect to different test instances $\mathbf{x}$.

Definition 11. 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:

$$benefit(V') = \sum_{\mathbf{x} \in I(V')} benefit(\mathbf{x}, A(V', \mathbf{x}))$$

$\square$

To illustrate the above definition, consider the resource dependency graph of Figure 1, where the attribute values for each test instance $\mathbf{x}_{a_i,b_j}$ are $hitcount(a_i)$, $hitcount(b_j)$, $hitcount(a_i \wedge b_j)$, and $dice(a_i, b_j)$, for $1 \leq i, j \leq 2$. Suppose now we want to compute $benefit(V')$ for $V' = \{hitcount(a_1), hitcount(b_1)\}$. Then $I(V') = \{\mathbf{x}_{a_1,b_1}, \mathbf{x}_{a_1,b_2}, \mathbf{x}_{a_2,b_1}\}$, and we have $A(V', \mathbf{x}_{a_1,b_1}) = \{hitcount(a_1), hitcount(b_1)\}$, $A(V', \mathbf{x}_{a_1,b_2}) = \{hitcount(a_1)\}$, and $A(V', \mathbf{x}_{a_2,b_1}) = \{hitcount(b_1)\}$. Thus, the benefit of V' is:

$$\begin{aligned} benefit(V') = & benefit(\mathbf{x}_{a_1,b_1}, \{hitcount(a_1), hitcount(b_1)\}) \\ & + benefit(\mathbf{x}_{a_1,b_2}, \{hitcount(a_1)\}) \\ & + benefit(\mathbf{x}_{a_2,b_1}, \{hitcount(b_1)\}) \end{aligned}$$

In this paper, we only use linear support vector machines. However, [Tan and Kan, 2010] shows that $E[mc(\mathbf{x})]$ and $E[mc(\mathbf{x} + A')]$ can be computed when a nonlinear kernel is used as well. Therefore, it is also possible to compute the proposed benefit function when the support vector machine uses an arbitrary kernel.

This completes the application of our resource acquisition framework to record matching problems where a support vector machine is used to classify whether record pairs are matches or otherwise.

Dataset	Short forms	Long forms	Matching pairs	Test instances	Vertices	Edges
GENOMES	307	307	307	23,409	117,657	140,760
	Source: http://www.ornl.gov/sci/techresources/Human_Genome/acronym.shtml Query: "$\langle title \rangle$" genome OR genomes					
DBLP	906	920	926	213,444	1,069,068	1,281,588
	Source: http://www.informatik.uni-trier.de/˜ley/db/conf/indexa.html Query: "$\langle title \rangle$" conference OR conferences OR workshop OR workshops					

Table 1: The evaluation datasets.

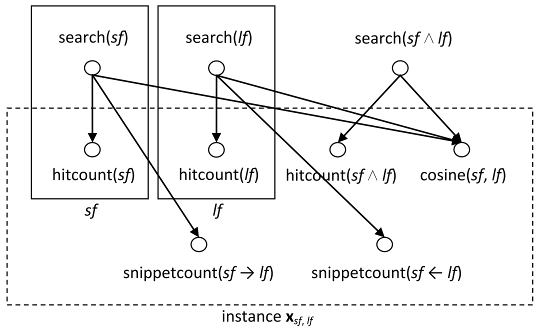


Figure 4: Structure of resource dependency graph used in the evaluations.

6 Evaluation

We evaluated our algorithm on an important real-world problem of matching short forms to long forms. Both people and systems are overwhelmed by the increasing number of short forms that appear in literature and metadata. Biologists struggle to cope with the large number of abbreviations related to DNA, proteins, and other biological terms. Maintainers of bibliographic databases need to detect and merge duplicate records, including publication venues where short forms like "KDD" can co-occur with long forms like "Knowledge Discovery and Data Mining". Fortunately, many abbreviations can be found on the web, albeit scattered across various web pages and not presented in a form that facilitates automated extraction. This makes the web a valuable resource for matching short forms to long forms.

This problem can be seen as a record linkage problem, with input lists SF and LF of short forms and long forms respectively, with the aim of finding which short forms $sf \in SF$ and which long forms $lf \in LF$ match. We used two datasets of different domains, GENOMES and DBLP, with very different characteristics. The GENOMES dataset consists of abbreviations that are commonly used in the human genomics domain, while the DBLP dataset consists of computer science conferences and workshops in the DBLP digital library. The GENOMES dataset, while smaller, is fairly noisy. It contains references to very different kinds of entities, and some long forms contain abbreviations in an inconsistent manner. As a result, the search engine results for GENOMES contain all kinds of sources and information. On the other hand, the DBLP dataset is manually constructed and fairly clean, but is much larger than GENOMES. Unlike the GENOMES dataset, we find a small number of short forms matching multiple long forms, and vice versa. The search engine results of DBLP is more homogeneous, mainly conference/workshop web sites and publication pages.

For each dataset, we split each of the short form and long form lists into two equal halves, and use one half for training the support vector machine classifier and the other half as test data. We construct an instance $\mathbf{x}_{sf,lf}$ for each short form sf and long form lf pair in the training data or test data, with attribute values extracted from the top-10 search engine results denoted by $search(sf)$, $search(lf)$, and $search(sf \wedge lf)$. The attribute values for an instance $\mathbf{x}_{sf,lf}$ are $hitcount(sf)$, $hitcount(lf)$, $hitcount(sf \wedge lf)$, $snippetcount(sf \rightarrow lf)$, $snippetcount(sf \leftarrow lf)$, and $cosine(sf, lf)$. The first three are the hit counts of sf, lf, and $sf \wedge lf$ respectively; the next two are the number of snippets from $search(sf)$ containing the lf string and the number of snippets from $search(lf)$ containing the sf string respectively [Tan et al., 2008]; and the last is computed by

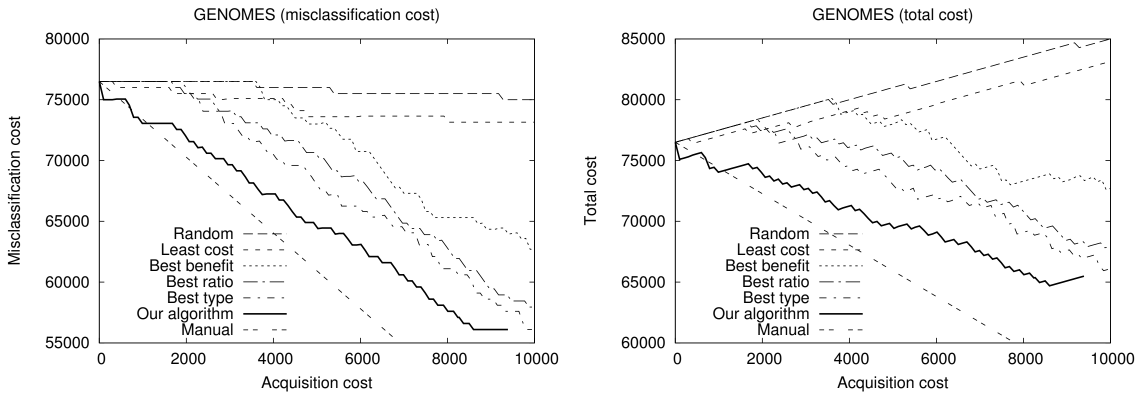


Figure 5: Results for GENOMES dataset.

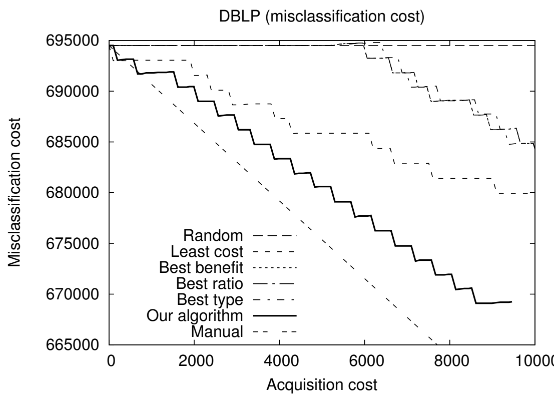
$\frac{hitcount(sf \wedge lf)}{\sqrt{hitcount(sf)hitcount(lf)}}$. These are are typical attributes used in related work. The resource dependency graph is depicted in Figure 4, and information about the graph is shown in Table 1. Note that the graph for DBLP has over 1 million vertices and 1 million edges. During training, five-fold cross-validation is performed on the training data to find the optimal support vector machine regularization parameter.

For our experiments, each attribute value and each search engine result acquisition cost is set to 1 and 10 respectively, as search engine queries take significantly longer to execute than extracting a attribute values. Misclassifying an instance that represents a mismatch as a match (false positive) incurs a set cost of 50, while false negatives incurs a cost of 500 and 1500 for GENOMES and DBLP respectively, owing to the relative ratios of matches to mismatches in these two datasets. Following [Ling et al., 2006], we use the total cost of acquisitions and misclassifications as our evaluation metric.

As our focus is to validate our acquisition framework and not on parameter tuning, we simply selected parameter values found to be effective. Additionally, we observed that our algorithm is not sensitive to changes in parameter values. For our Tabu search based algorithm, we use a static Tabu tenure of 7 (used in many other Tabu search applications), terminate our algorithm when 20 consecutive iterations have occurred without an increase in our objective value, use a propagation factor λ of 0.5, and an aggregation factor β of 0.1. 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, and then we execute the algorithm 100 times. Each time, the algorithm will be executed with a budget of 100, and result in some unacquired vertices being acquired. At the end of each execution, we recorded the cumulative acquisition cost of all the vertices acquired so far, as well as the misclassification cost of all the instances, and the total cost of the previous two, giving one data point. Eventually, all the data points for the algorithm are then plotted on two charts, one showing the misclassification cost over acquisition cost, and the other showing total cost over acquisition cost. These charts allow us to compare the performance of the different algorithms.



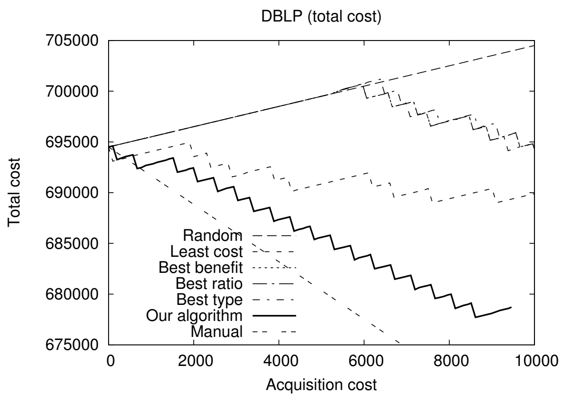


Figure 6: Results for DBLP dataset.

We also include in these charts a line labeled **Manual**, which estimates the best possible performance for any algorithm. This manual process is performed with the knowledge of which attribute value acquisitions will allow which test instances to be correctly classified by the support vector machine. We manually acquired vertices that allowed all instances to be correctly classified while keeping acquisition costs as low as possible, taking advantage of an observation that the required acquisitions for many of the instances fall into a few patterns. This produces one data point, which we joined with the data point where no resource acquisitions have taken place. Naturally, this manual process, guided by expert domain knowledge, will outperform any automated algorithm.

The charts for the GENOMES dataset and the DBLP dataset are shown in Figure 5 and Figure 6 respectively. From these charts, it can be seen that our algorithm consistently and significantly outperforms all the baseline algorithms we evaluated against. Our algorithm is able to reduce the total cost of acquisitions and misclassifications as more vertex acquisitions are made.

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 at or above 1000 in intervals of 100. For the GENOMES dataset, our algorithm makes an average improvement of 49.7% over best type algorithm, which is the second best. For the DBLP dataset, our algorithm makes an average improvement of 49.5% over the least cost algorithm, which is the second best. The average improvement is nearly 50%, a very significant improvement. Importantly, our algorithm consistently holds as the best algorithm over the entire range of acquisition costs plotted in both figures.

We now consider the test instances that are correctly classified when our acquisition algorithm is used, and compare it with the case when all the attribute values of these instances are acquired. For both datasets, we found that our algorithm acquires about 10% of the $search(\cdot)$ vertices, saving about 90% of the search engine queries. This is because each $search(sf)$ and $search(lf)$ services a large number of test instances, whereas each $search(sf \wedge lf)$ services only a single test instance. As many of the instances do not require the information in $search(sf \wedge lf)$ to be classified correctly, our algorithm mostly made search engine calls of the form $search(sf \wedge lf)$, resulting in significantly lesser 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.

7 Conclusion

Web information is increasingly used for solving various kinds of problems, including record matching problems, simply because the web provides much more information and much better frequency-based statistics than any offline corpus or database. However, acquiring web-based resources is slow and can incur other access costs, but existing work has largely ignored this issue. The challenge is to decide what data to download, in order to be scalable. Unfortunately, existing work that deals with selective acquisition of resources fails to put sufficient emphasis on the hierarchical nature of resource acquisitions. This makes hierarchical cost-sensitive acquisition of resources an important area of research.

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 scenarios, as long as a benefit function can be supplied. In the context of web resource acquisition, we showed that the resource acquisition graphs for record matching problems have a unique structure. This requires us to devise a search algorithm that can answer the challenges presented by such resource acquisition graphs. We also proposed a benefit function when the classifier is the support vector machine, and evaluate our proposed acquisition algorithm for this classifier. Our evaluation on a short form to long form matching problem shows that our acquisition algorithm significantly outperforms a number of baseline algorithms.

References

[Bollegala et al., 2007] Bollegala, D., Matsuo, Y., and Ishizuka, M. (2007). Measuring semantic similarity between words using web search engines. In *International Conference on World Wide Web (WWW)*, pages 757–766.

[Cimiano et al., 2005] Cimiano, P., Ladwig, G., and Staab, S. (2005). Gimme' the context: Context-driven automatic semantic annotation with C-PANKOW. In *International Conference on World Wide Web (WWW)*, pages 332–341.

[Davis et al., 2006] Davis, J. V., Ha, J., Rossbach, C. J., Ramadan, H. E., and Witchel, E. (2006). Cost-sensitive decision tree learning for forensic classification. In *European Conference on Machine Learning (ECML)*, pages 622–629.

[Elmacioglu et al., 2007] Elmacioglu, E., Kan, M.-Y., Lee, D., and Zhang, Y. (2007). Web based linkage. In *ACM International Workshop on Web Information and Data Management (WIDM)*, pages 121–128.

[Elmagarmid et al., 2007] Elmagarmid, A. K., Ipeirotis, P. G., and Verykios, V. S. (2007). Duplicate record detection: A survey. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 19(1):1–16.

[Glover, 1990] Glover, F. (1990). Tabu search: A tutorial. *Interfaces*, 20(4):74–94.

[Goiser and Christen, 2006] Goiser, K. and Christen, P. (2006). Towards automated record linkage. In *Australasian Conference on Data Mining and Analystics (AusDM)*, pages 23–31.

[Kalashnikov et al., 2008] Kalashnikov, D. V., Nuray-Turan, R., and Mehrotra, S. (2008). Towards breaking the quality curse: a web-querying approach to web people search. In *ACM Conference on Research and Development in Information Retrieval (SIGIR)*, pages 27–34.

[Kanani and McCallum, 2007] Kanani, P. and McCallum, A. (2007). Resource-bounded information gathering for correlation clustering. In *Computational Learning Theory (COLT)*, pages 625–627.

[Ling et al., 2006] Ling, C. X., Sheng, V. S., and Yang, Q. (2006). Test strategies for cost-sensitive decision trees. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 18(8):1055–1067.

[Matsuo et al., 2006] Matsuo, Y., Mori, J., Hamasaki, M., Ishida, K., Nishimura, T., Takeda, H., Hasida, K., and Ishizuka, M. (2006). POLYPHONET: An advanced social network extraction system from the web. In *International Conference on World Wide Web (WWW)*, pages 397–406.

[Oh and Isahara, 2008] Oh, J.-H. and Isahara, H. (2008). Hypothesis selection in machine transliteration: A web mining approach. In *International Joint Conference on Natural Language Processing (IJCNLP)*, pages 233–240.

[Platt, 2000] Platt, J. (2000). Probabilistic outputs for support vector machines and comparison to regularized likelihood methods. In *Advances in Large Margin Classifiers*, pages 61–74.

[Saar-Tsechansky et al., 2009] Saar-Tsechansky, M., Melville, P., and Provost, F. (2009). Active feature-value acquisition. *Management Science*, 55(4):664–684.

[Sahami and Heilman, 2006] Sahami, M. and Heilman, T. D. (2006). A web-based kernel function for measuring the similarity of short text snippets. In *International Conference on World Wide Web (WWW)*, pages 377–386.

[Smalheiser and Torvik, 2009] Smalheiser, N. R. and Torvik, V. I. (2009). Author name disambiguation. *Annual Review of Information Science and Technology (ARIST)*, 43.

[Tan et al., 2008] Tan, Y. F., Elmacioglu, E., Kan, M.-Y., and Lee, D. (2008). Efficient web-based linkage of short to long forms. In *International Workshop on the Web and Databases (WebDB)*.

[Tan and Kan, 2010] Tan, Y. F. and Kan, M.-Y. (2010). Cost-sensitive attribute value acquisition for support vector machines. Technical Report TRB3/10, School of Computing, National University of Singapore.

[Tan et al., 2006] Tan, Y. F., Kan, M.-Y., and Lee, D. (2006). Search engine driven author disambiguation. In *ACM/IEEE Joint Conference on Digital Libraries (JCDL)*, pages 314–315.

[Winkler, 2006] Winkler, W. E. (2006). Overview of record linkage and current research directions. Technical Report RRS2006/02, U.S. Bureau of the Census.