ACADEMIC PRESS
Available at www.ComputerScienceWeb.com
POWERED BY SCIENCE DIRECT
Journal of Computer and System Sciences 66 (2003) 809–851
JOURNAL OF COMPUTER AND SYSTEM SCIENCES
http://www.elsevier.com/locate/jcss

Querying websites using compact skeletons

Anand Rajaraman[a,*] and Jeffrey D. Ullman[b,1]

[a] *Cambrian Ventures, 201 San Antonio Circle, Mountain View, CA 94040, USA*
[b] *Department of Computer Science, Stanford University, Stanford, CA 94301, USA*

Received 6 September 2001; revised 13 July 2002

Abstract

Several commercial applications, such as online comparison shopping and process automation, require integrating information that is scattered across multiple websites or XML documents. Much research has been devoted to this problem, resulting in several research prototypes and commercial implementations. Such systems rely on wrappers that provide relational or other structured interfaces to websites. Traditionally, wrappers have been constructed by hand on a per-website basis, constraining the scalability of the system. We introduce a website structure inference mechanism called *compact skeletons* that is a step in the direction of automated wrapper generation. Compact skeletons provide a transformation from websites or other hierarchical data, such as XML documents, to relational tables. We study several classes of compact skeletons and provide polynomial-time algorithms and heuristics for automated construction of compact skeletons from websites. Experimental results show that our heuristics work well in practice. We also argue that compact skeletons are a natural extension of commercially deployed techniques for wrapper construction.

1. Introduction

Several commercial applications, such as online comparison shopping and process automation, require integrating information that is scattered across multiple websites or hierarchically structured documents (e.g., in XML). Much research has been devoted to this problem, resulting in several research prototypes and commercial implementations (e.g., [15,20,25,35]). Such systems rely on components called *wrappers* that provide relational or other structured interfaces to websites. Traditionally, wrappers have been constructed by hand on a per-website basis, because

*Corresponding author.
E-mail addresses: anand@cambrianventures.com (A. Rajaraman), ullman@cs.stanford.edu (J.D. Ullman).
[1] Work partially supported by NSF Grant IIS-9811904.

0022-0000/03/$ - see front matter © 2003 Elsevier Science (USA). All rights reserved.
doi:10.1016/S0022-0000(03)00029-1

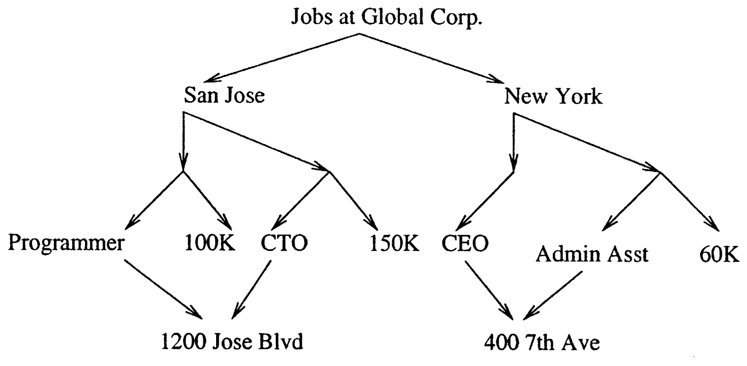
each website uses its own data vocabulary, organization, and access mechanisms. Constructing these wrappers has been a major bottleneck preventing data-integration systems from scaling; most systems today integrate tens rather than thousands of websites.

We introduce a website structure inference mechanism called *compact skeletons* that is a step in the direction of automated wrapper construction. We focus on an important special case: given a single relation scheme, we need to piece together information elements scattered across a website to constitute the relation. For example, suppose our application integrates job listings from websites on the Internet (FlipDog.com [34] is such an application). Let us say that each job listing has a job title T, a salary S, and an address A for candidates to send their resumes. The relation scheme then is $R(TSA)$.

Fig. 1 shows a portion of a corporate website that lists job openings. For the moment, assume that each node corresponds to a web page where we have eliminated all the irrelevant text, leaving behind only the data element of interest. The website is oriented towards a human reader, and to a person it is fairly obvious what the tuples in the relation are. However, even this extremely simple website illustrates some of the difficulties encountered by a program trying to materialize the relation R, such as superfluous information (the job location) and incomplete information (the CEO's compensation package is negotiable).

Given a website such as that in Fig. 1 and a target relation scheme, we break up the problem of constructing a wrapper into three subproblems:

1. Identifying the data elements in the scheme, such as addresses, job titles, and salaries.
2. Deducing the principles that have been followed by the person who put together the website (in effect, "reverse engineering" the website).
3. Constructing the relation corresponding to the website. In practice, we need only materialize the portion of the relation that is relevant to the query at hand.



Title	Salary	Address
Programmer	100K	1200 Jose Blvd
CTO	150K	1200 Jose Blvd
CEO	$\perp$	400 7th Ave
Admin Asst	60K	400 7th Ave

Fig. 1. Data graph of a website advertising job openings and the corresponding relation.

This three-step approach is commonly used in the industry. For example, Whizbang! Labs [35] calls it the "C^4 technique" (where the 4 C's are *crawl*, *classify*, *capture*, *and compile*; we do not include crawling in our taxonomy), while Junglee's Virtual Database Management System [20] has components called *extractors*, *wrappers*, and *mappers* corresponding to these three steps.

A simple way to tackle problem (1) is to use a library of patterns (such as regular expressions). There are several approaches to constructing such patterns: "by hand" by studying several examples [19]; machine learning techniques; and more novel pattern extraction techniques [7].

Our work deals with problems (2) and (3). Once we have identified the patterns of interest on the pages of a website, we can model the website as a directed graph with data elements at the nodes. We assume that the domains of the schema attributes are pairwise disjoint, so that we can unambiguously associate each data value with it corresponding attribute. There are (unlabeled) arcs in the graph corresponding both to structure within a web page (in the case where we identify multiple data elements within a web page) and to hyperlinks between pages. We call such a graph a *data graph*; Fig. 1 is an example. In the rest of the paper, we model websites as data graphs. Data graphs have been used extensively in the literature to model semistructured data, e.g., in [1,6, 8–11,26,30].

Compact skeletons are labeled trees that function as transformations between data graphs and relations. Intuitively, a compact skeleton describes the hierarchical layout of the corresponding website: for example, the IBM site groups jobs first by division (D), and each listing includes a job id (I), a job title (T), a job category (C), and the state where the job is in (S). The job title is hyperlinked to details about the job (J) and an address to send resumes to apply for the job (A). This hierarchy is captured by the corresponding compact skeleton, shown in Fig. 15(a). Compact skeletons are a natural extension of Junglee's Site Description Language (SDL) [19], which has been used to construct thousands of wrappers for Junglee's VDBMS [20]. We describe the relationship between SDL and compact skeletons in Section 9.

The rest of this paper is organized as follows. In Section 2 we introduce compact skeletons and analyze the properties of *perfect compact skeletons* (PCS), which apply when the data graph has complete information. In Section 3 we relax the completeness condition and introduce *partially perfect compact skeletons* (PPCS) that apply when the data graph has incomplete information, corresponding to null values in relations as in Fig. 1. For a given data graph, we show that the PCS is unique but the PPCS is not; we introduce the notions of *minimal* and *maximal* PPCS that provide upper and lower bounds on the relation associated with the data graph. We describe polynomial-time algorithms to compute the PCS and the minimal and maximal PPCS. In Section 4 we present algorithms for querying websites given a compact skeleton; a special case is to materialize the entire relation corresponding to the website.

Real-life websites often contain *noise* (i.e., superfluous information) in addition to incomplete information. In Sections 5 and 6 we study *best-fit skeletons* (BFS) that apply in such cases. It turns out that computing the BFS is an NP-complete problem. We examine two simple polynomial-time heuristics, the *greedy* and the *weighted greedy*. Experimental results show that the heuristics work well in practice. In Section 7 we discuss some of the practical issues that arise when applying the skeleton technique, such as websites that use form inputs.

The skeletons we consider in Sections 2–7 are restricted to be labeled trees. Section 8 extends the theory to *graph skeletons*, where we permit skeletons to be arbitrary labeled graphs. We show that the PCS remains unique and provide a non-deterministic polynomial-time algorithm to compute it. Section 9 examines related work, and Section 10 concludes with some directions for further investigation.

2. Data graphs and skeletons

2.1. Data graphs

We model websites using data graphs as shown in Fig. 1. We restrict our attention to data graphs that are DAGs; we relax this restriction in Section 8 when we discuss generalized skeletons. For simplicity, we assume that each node of the graph has at most one information element (in our example, an instance of A, T, or S). In practice, if a page contains multiple elements of information, we create new nodes corresponding to each element and add an arc from the node corresponding to the page to these newly created nodes. We also ignore information in nodes other than that corresponding to the schema attributes. We assume, without loss of generality, that if a data graph includes a node, then there is some information element that is reachable from the node. In particular, this assumption means that a node with no outgoing arc must contain an information element.

Let X be the set of attributes in our schema. For attribute A in X, the domain of A, denoted by $Dom(A)$, is the (possibly infinite) set of all possible values that can appear in column A of relations over the schema. For simplicity, we assume that domains of attributes in the schema are pairwise disjoint. We use uppercase letters $A, B, C, \ldots$ for attribute names and corresponding lowercase letters (often with subscripts) for data values corresponding to those attributes, e.g., $a_1, a_2, \ldots \in Dom(A)$.

Given a graph G, we use V_G and E_G to denote respectively the node set and the arc set of G. If v is in V_G, we denote by $val(v)$ the value at node v. If there is no value at v, then $val(v) = \perp$, where $\perp$ is a special value that is not in the domain of any attribute. If $val(v)$ is in $Dom(A)$, we say that A is the attribute at node v, written as $attr(v) = A$. We use the convention that if $val(v) = \perp$, then $attr(v) = \perp$.

2.2. Skeletons

Suppose we are given a relation R over attribute set X, and we wish to construct a data graph that incorporates the contents of R. One way to do so is to work incrementally, using the following imaginary process. We construct the data graph corresponding to the first tuple in R, with some nodes and edges. The structure formed by these nodes and edges depends on how we intend to lay out the web site. We then add nodes and edges corresponding to the second tuple, and continue this process until we have the data graph corresponding to the entire relation. If we have been consistent about this process (e.g., as is increasingly common, the website was generated by a program), the manner in which we interconnected the values of the attributes corresponding to the first tuple will be identical to that for the second tuple, and so on. We formalize this notion as follows.

A *skeleton* K is a tree some of whose nodes are labeled with attribute names from X such that each attribute labels exactly one node. If node v in V_K is labeled with attribute A, we say that

$attr(v) = A$. There may be nodes of the skeleton that mention no attributes at all. If i is such a node, we say $attr(i) = \perp$. For any tuple t in R, the tree $K(t)$ is obtained by constructing a tree isomorphic to K, replacing each attribute with the corresponding value from t. For tuples t_1 and t_2, nodes u_1 and u_2 in $K(t_1)$ and $K(t_2)$ are *similar* if they correspond to the same node of K, and $val(u_1) = val(u_2)$. We now describe the incremental process to construct a data graph for R. We assume without loss of generality that either the root of K is unlabeled, or the roots of all the $K(t_i),\ t_i \in R$, have the same value.

- For the first tuple, t_1, the data graph $G_1 = K(t_1)$.
- Suppose we have constructed G_{r-1}, corresponding to the first $r-1$ tuples of R. To construct G_r, we add the nodes and edges of $K(t_r)$ to G_{r-1}, identifying the root of $K(t_r)$ with the root of G_{r-1}. In addition, choose some arbitrary subset of nodes from $K(t_r)$ and identify each node with a similar node from some $K(t_i)$, where $1 \leqslant i < r$. When both ends of a pair of edges get identified, identify the edges.

We may view a skeleton as a transformation from a relation to a data graph. Given a relation R and a skeleton K, there are several data graphs corresponding to the two, depending on which sets of nodes and edges we choose to identify. If G is any such data graph, we write $R \xrightarrow{K} G$. Note that even though a skeleton is a tree, data graphs constructed using the skeleton need not be trees; tree skeletons can give rise to DAG data graphs. We can generalize the construction process to allow for data graphs with more than one root. All our results generalize to such data graphs.

2.3. Compact skeletons

Given a data graph G and a skeleton K, let ϕ be an isomorphism between K and some subgraph G' of G. We say that ϕ is an *overlay* from K to G if the following conditions are satisfied:

- ϕ maps the root of G to the root of K.
- Suppose u is a node of G' and v is the corresponding node in K. Then $attr(u) = attr(v)$.

If ϕ is such an overlay, we say that ϕ *includes* all the nodes and edges in G'. For node u in G', if $attr(u) = A$ and $val(u) = a$, then we say that $\phi(A) = a$. The tuple $t = \phi(X)$ corresponding to the overlay ϕ is defined in the natural manner as the list of the values $\phi(A)$ for all A in X. The relation $R(G, K)$ induced by the skeleton K is the set of all tuples t such that there is an overlay ϕ from K to G with $\phi(X) = t$. We say that a skeleton K is *perfect* for data graph G if for every arc e in E_G, there is at least one overlay that includes e.

Example 2.1. Fig. 2 shows a simple data graph G and two perfect skeletons over the attribute set ATS; let us call Fig. 2(b) K_1 and Fig. 2(c) K_2. In Fig. 2, there are 3 possible overlays of the skeleton K_1, and the corresponding relation is given by

$$R_1 = R(G, K_1) = \{a_1 t_1 s_1, a_1 t_2 s_2, a_2 t_3 s_3\}.$$

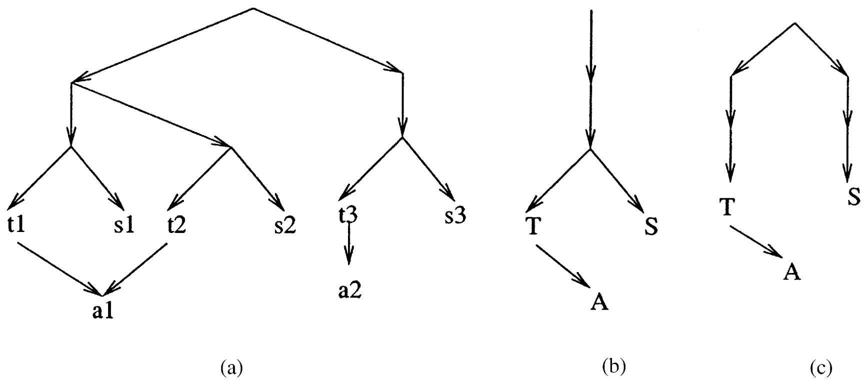


Fig. 2. A data graph and two skeletons corresponding to it.

There are 4 possible overlays of the skeleton K_2, and the corresponding relation is

$$R_2 = R(G, K_2) = \{a_1t_1s_3, a_1t_2s_3, a_2t_3s_1, a_2t_3s_2\}.$$

Note that ϕ is an isomorphism, so it is not possible for two nodes of the skeleton to correspond to the same node of the data graph. This observation explains why we do not find tuples such as $a_1t_1s_1$ in R_2.

Given a data graph, there may be several perfect skeletons corresponding to it, each inducing a different relation. In many cases, these relations do not conform to our common-sense notion of the relation corresponding to the data graph. In Example 2.1, the relation corresponding to the first skeleton seems intuitively to be "right" while that corresponding to the second skeleton does not. It appears that the second skeleton violates some notion of locality: we expect information elements that are "closer" to each other to combine to produce tuples in favor of combining elements that are "far away" from each other. More formally, it can be verified that while $R_1 \overset{K_1}{\rightarrow} G$, it is not true that $R_2 \overset{K_2}{\rightarrow} G$; in fact, it can be verified that there is no relation R satisfying $R \overset{K_2}{\rightarrow} G$.

We now formalize this notion. A *compact skeleton* K for data graph G is a skeleton that satisfies the following condition: for every node u in G, there is a node v in K such that in every overlay ϕ from K to G in which node u participates, u is mapped to v. We call v the *K-image* of u, denoted by $image_K(u) = v$. A skeleton K is a *perfect compact skeleton* (PCS) for a data graph G if K is compact for G and K is perfect for G.

Example 2.2. The skeleton in Fig. 2(b) is a compact skeleton, while that in Fig. 2(c) is not. To verify the latter, consider the overlays that result in the tuples $a_1t_1s_3$ and $a_2t_3s_1$: the children of the root of G get mapped to different nodes of the skeleton by the two overlays.

Not every data graph has a perfect compact skeleton (PCS). However, every data graph constructed in the manner described in the foregoing section has a PCS. In addition, perfect compact skeletons possess two desirable properties: locality and uniqueness. The locality property reflects the observation that large websites are typically constructed so that local fragments make sense independent of the global picture. The following sequence of lemmas, leading up to Theorem 2.1, formalizes our results. We use the notation that if $u \in V_G$, then G_u is the subgraph of G that is reachable from u.

Lemma 2.1. *If $R \overset{K}{\rightarrow} G$, then K is a perfect compact skeleton (PCS) for G.*

Proof. It is easy to verify that the construction process for G can never create a cycle, so G must be a DAG. We use induction on the distance of node u in G from a sink (a node with no outgoing arc) to prove that every overlay maps u to the same node of K. For the basis, let u be a sink in G. By assumption, u contains an information element, and any overlay must map u to the unique node in K labeled with the corresponding attribute.

Suppose that every node at distance less than d from a sink is mapped to a unique node of K in every overlay, and let $u \in V_G$ be at distance d from a sink. There is at least one node $v \in V_G$ such that $(u, v) \in E_G$ and v is at distance less than d from a sink. By hypothesis, every overlay maps v to a unique node $x \in K$. It follows that every overlay must map u to the unique node that is the parent of x in K. It follows by induction that K is a compact skeleton for G.

To see that K is perfect, let $R = \{t_1, \dots, t_n\}$ and consider the overlays corresponding to $K(t_1), \dots, K(t_n)$. Clearly every edge of G must have originated from some $K(t_i)$, $1 \leqslant i \leqslant n$, and so this set of overlays covers every edge of G. It follows that K is a PCS for G. □

Lemma 2.2. *If K is a PCS for G and $u \in V_G$, then there is a subtree K' of K that is a PCS for G_u, and the distance of the root of K' from the root of K is equal to the distance of u from the root of G.*

Proof. For each node $u \in V_G$ with $image_K(u) = x$, we show that K_x is a PCS for G_u by induction on the distance of u from a sink. The assertion clearly holds for the sinks of G. Suppose the assertion is true for nodes at distance less than d from a sink, and let $u \in V_G$ be at distance d from a sink. Let $image_K(u) = x$, let $y_1, \dots, y_m$ be the children of x in K, and let $v_1, \dots, v_n$ be the children of u in G. It follows that for each v_i, $1 \leqslant i \leqslant n$, there is a y_j, $1 \leqslant j \leqslant m$, such that $image_K(v_i) = y_j$, and by induction hypothesis K_{y_j} is a PCS for G_{v_i}. Let e be an edge in G_u. We consider two cases to show that for every edge e in G_u, there is an overlay of K_x that includes e:

Case 1: $e = (u, v_i)$, $1 \leqslant i \leqslant n$. Since K is a PCS for G, there is an overlay ϕ of K on G that includes e, and maps x to u. The restriction of ϕ to the subtree K_x of K is an overlay of K_x on G_u that includes e.

Case 2: e is in G_{v_i}, $1 \leqslant i \leqslant n$. Without loss of generality, let $image_K(v_i) = y_1$. Since K_{y_1} is a PCS for G_{v_i}, there is an overlay ϕ_1 of K_{y_1} on G_{v_i} that includes e. For $2 \leqslant j \leqslant m$, let ϕ_j be any overlay of K_{y_j} on some G_{v_l} with $image_K(v_l) = y_j$. Construct an overlay ϕ of K_x on G_u as follows: $\phi(x) = u$, and $\phi(y) = \phi_j(y)$ for $y \in K_{y_j}$, $1 \leqslant j \leqslant m$. Then ϕ is an overlay of K_x on G_u that includes e.

It follows by induction that for each node $u \in V_G$ with $image_K(u) = x$, K_x is a PCS for G_u. Since K is a tree, there is a unique path of length l (say) from the root of K to node x. If $image_K(u) = x$, it follows by induction on l that all paths from the root of G to u must be of length l. We therefore have the lemma. □

Lemma 2.3. *Every data graph G has either a unique PCS or no PCS.*

Proof. The proof is by induction on the number of nodes in G. We can restrict ourselves to DAGs since G has no PCS if it is not a DAG. For the basis, if G has a single node with an information element, then it clearly has a unique PCS. Suppose the lemma is true for DAGs of fewer than n nodes, and let G be a DAG with n nodes. Let r be the root of G and let $u_1, \ldots, u_m$ be the children of u. By hypothesis, each of $G_{u_1}, \ldots, G_{u_m}$ satisfies the lemma. If any one of the G_{u_i} has no PCS, it follows from Lemma 2.2 that G has no PCS, and so G satisfies the lemma. Otherwise let $K_1, \ldots, K_m$ denote the unique PCS respectively of $G_{u_1}, \ldots, G_{u_m}$. From Lemma 2.2, in any PCS K of G, each of $K_1, \ldots, K_m$ must be a subtree of the root. There is only one tree K that can satisfy this property: the root of K is labeled with $attr(r)$, and the children of the root are the unique K_i, $1 \leqslant i \leqslant m$ (formally, choose the set of K_i such that $K_j \neq K_i$ for $j < i$). It follows that either K is the unique PCS for G or G has no PCS. □

Lemmas 2.1–2.3 imply the following theorem.

Theorem 2.1. *For any relation R, data graph G, and skeleton K:*

- *If $R \xrightarrow{K} G$, then K is the unique PCS for G.*
- *Conversely, if K is a PCS for G, there is a relation R such that $R \xrightarrow{K} G$.*
- *If K is a PCS for G and $u \in V_G$, then there is a subtree K' of K that is a PCS for G_u.*

2.4. Computing the PCS

Given a data graph G, there is a simple algorithm to determine if it has a PCS and to compute one if it exists. If $u \in V_G$, the *attribute set* associated with u, denoted by $attrset(u)$, is the set of all attributes whose values appear in nodes reachable from u. In what follows, for trees T_1 and T_2, $T_1 \equiv T_2$ if T_1 and T_2 are isomorphic.

Algorithm. ComputePCS

```
1. For each sink $u$ of $G$, $T_u$ is the single-node skeleton labeled with $attr(u)$.
2. Process $G$ bottom-up by successive elimination of sinks. Suppose $u$ is the current node. Let
   $v_1, \ldots, v_m$ be the children of $u$, and let $X_i = attrset(v_i)$ and $T_i = T_{v_i}$ for $1 \leqslant i \leqslant m$. Process $u$ as
   follows:
    (i) If there is a pair $i, j$ such that $X_i \cap X_j \neq \emptyset$ and $T_i \not\equiv T_j$, then $G$ has no PCS.
   (ii) If $attr(u) \neq \bot$ and $attr(u) \in X_i$ for some $i$, $1 \leqslant i \leqslant m$, then $G$ has no PCS.
  (iii) Construct $T_u$ as follows: the root of $T_u$ is a node labeled with $attr(u)$; the subtrees of the
        root are given by $\{T_i \mid \forall j < i, T_i \not\equiv T_j\}$.
3. Let $r$ be the root of $G$. Then $T_r$ is the unique PCS for $G$.
```

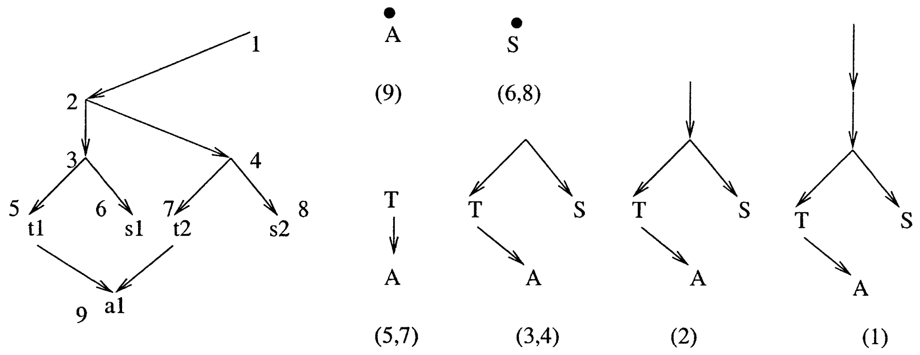


Fig. 3. Running Algorithm ComputePCS.

Example 2.3. Fig. 3 shows a portion on the website from Fig. 2 with nodes labeled using numeric identifiers, and the trees constructed by Algorithm *ComputePCS* after processing each node of the website. The tree that results after node 1 (the root of the website) is processed is the unique PCS for the website.

Theorem 2.2. *For any data graph G, Algorithm ComputePCS either computes the PCS for G or determines that G has no PCS, and runs in time $O(km|V_G|)$, where k is the number of attributes in the relation scheme and m is the number of nodes in the PCS of the largest subgraph of G that has a PCS.*

Proof. We prove by induction on the order the nodes are processed that after node u is processed, either T_u is the PCS for G_u or G_u has no PCS; correctness then follows from Lemma 2.2. For the basis, note that the assertion is true for the sinks of G. Suppose the assertion holds after several iterations and we are currently processing node u. The cases below correspond to the cases of Step 2 in the algorithm.

1. Suppose there is a pair of children of u such that $X_i \cap X_j \neq \emptyset$ and $T_i \not\equiv T_j$. We must prove that G_u has no PCS. Suppose to the contrary that G_u has a PCS T; Lemmas 2.2 and 2.3 imply that the root of T has subtrees $T_i' \equiv T_i$ and $T_j' \equiv T_j$. Let $A \in X_i \cap X_j$. Since A appears exactly once in T, it follows that if A appears both in T_i' and T_j' where neither T_i' is not a subtree of T_j' and T_j' is not a subtree of T_i', then $T_i' = T_j'$, a contradiction.

2. Suppose $attr(u) \neq \perp$ and $attr(u) \in X_i$ for some node v_i. There is some node v in T_i labeled with $attr(u)$. If G_u has a PCS T, its root must be labeled by $attr(u)$, and since T must contain a subtree $T_i' \equiv T_i$ (by Lemma 2.2), there is more than one node in T labeled by the same attribute, a contradiction.

3. A case analysis similar to that used in the proof of Lemma 2.2 shows that T_u is a PCS for G_u.

Correctness thus follows by induction. To obtain the complexity bound, notice that each T_i must have at least one leaf labeled with some attribute, and so each X_i is nonempty. There can be at most k distinct nonempty values of X_i such that $X_i \cap X_j = \emptyset$. Thus there are at most k distinct trees among the T_i in Step 2. By a careful implementation, each tree T_i needs to be compared for equality with at most k other trees. Each comparison takes time proportional to the sum of the sizes of the trees, which is $O(m)$. The construction of T_u also takes time $O(m)$. Thus each iteration of Step 2 takes time $O(km)$, and there are at most $|V_G|$ iterations of Step 2, giving the complexity bound. □

3. Partially perfect compact skeletons

In the real world, it often happens that a data graph has no PCS because it has incomplete information. For example, the data graph in Fig. 1 is missing a salary for the CEO, and has no PCS. Incomplete information can arise even when the relation R underlying a data graph G is complete. It may happen that our algorithms for identifying data values in the website produce false negatives e.g., we may not identify a job title because it does not match our patterns.

We model missing information in a data graph as follows. We start from a relation R that has nulls, constructing data graph G using skeleton K in the usual manner. Given tuple t, possibly containing nulls, the graph $K(t)$ is obtained by replacing attribute names in K with the corresponding values from t and then deleting redundant nodes, where a node is redundant if no node with a non-null value is reachable from it. With this modification, the procedure for constructing a data graph from a relation remains the same as before, and we use the same notation $R \xrightarrow{K} G$.

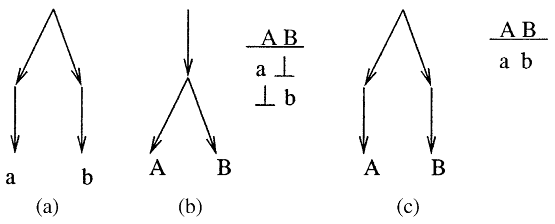
For data graphs with incomplete information, we relax the notion of PCS as follows. Let G be a data graph and K a skeleton over the same schema. Let ϕ be an isomorphism between a connected subgraph K' of K and a subgraph G' of G. We say that ϕ is a *partial overlay* from K to G if the following conditions are satisfied:

- ϕ maps the root of K to the root of G.
- Suppose u is a node of G and v is the corresponding node of K'; then $attr(u) = attr(v)$.

If ϕ is such a partial overlay, we say that ϕ *includes* the nodes and edges of G', and that the subgraph K' *covers* the nodes and edges of G'. The tuple t corresponding to ϕ is defined in the natural manner as the list of the information elements in G', padded with nulls in those attributes of K that do not appear in K'. We call ϕ a *minimal partial overlay* if there is no partial overlay ϕ' that uses a strict subset of the nodes in K and derives the same tuple t.

The relation $R(G,K)$ is obtained by taking the union of the tuples produced by any partial overlay of K on G and then performing tuple subsumption, defined in the usual manner. We say that tuple t_1 *subsumes* tuple t_2 if the non-null attribute values in t_1 are the same as those in t_2; the subsumption is *strict* if t_1 has at least one additional non-null value. In $R(G,K)$ we eliminate a tuple t if there is another tuple t' that strictly subsumes t.

We define K to be *partially compact* for G if for each node u in V_G, there is a node v in V_K such that every minimal partial overlay maps u to v, and every node of K is mapped by some minimal partial overlay. The latter condition eliminates skeletons containing spurious nodes that do not appear in any minimal partial overlay, but which technically would still be compact otherwise. A skeleton K is *partially perfect* for G if for every edge e in E_G, there is a partial overlay of K that



A	B
a	⊥
⊥	b

A	B
a	b

Fig. 4. A data graph and two partially perfect compact skeletons.

includes e. Combining these two definitions, K is a *partially perfect compact skeleton* (PPCS) for G if K is partially perfect for G and partially compact for G.

The following sequence of lemmas leads up to Theorem 3.1, which generalizes Theorem 2.1 for partially perfect compact skeletons. Note that while the locality property holds for partially perfect compact skeletons, uniqueness does not (Fig. 4).

Lemma 3.1. *For any relation R, possibly containing null values, and data graph G, if* $R \xrightarrow{K} G$, *then K is a PPCS for R. Conversely, if K is a PPCS for a data graph G, then there is a relation R such that* $R \xrightarrow{K} G$.

Proof. The proof is very similar to that of Lemma 2.1 Suppose $R \xrightarrow{K} G$. G must be a DAG, and we use induction on the distance of node u in G from a sink (a node with no outgoing arc) to prove that every minimal partial overlay maps u to the same node of K. For the basis, let u be a sink in G. By assumption, u contains an information element, and any partial overlay must map u to the unique node in K labeled with the corresponding attribute.

Suppose that every node at distance less than d from a sink is mapped to a unique node of K in every minimal partial overlay, and let $u \in V_G$ be at distance d from a sink. We consider two cases:

Case 1: If $val(u) \neq \perp$, every minimal partial overlay must map u to the unique node in K labeled with $attr(u)$.

Case 2: If $val(u) = \perp$, any minimal partial overlay that includes u also includes at least one node $v \in V_G$ such that $(u, v) \in E_G$ and v is at distance less than d from a sink (else we could delete u from the overlay and derive the same tuple). By hypothesis, every overlay maps v to a unique node $x \in K$. It follows that any minimal partial overlay that includes u must map u to the unique node that is the parent of x in K.

It follows by induction that K is a partially compact skeleton for G. To see that K is partially perfect, let $R = \{t_1, \ldots, t_n\}$ and consider the partial overlays corresponding to $K(t_1), \ldots, K(t_n)$. Clearly, every edge of G must have originated from some $K(t_i)$, $1 \leqslant i \leqslant n$, and so this set of partial overlays covers every edge of G. It follows that K is a PPCS for G.

For the converse, if K is a PPCS for G, set $R = R(G, K)$ to obtain a relation R such that $R \xrightarrow{K} G$. □

Lemma 3.2. *Let K be a PPCS for a data graph G, and let* $u \in V_G$ *such that* $image_K(u) = x$. *Then* K_x *is a PPCS for* G_u.

Proof. To show that K_x is partially compact for G_u, let ψ be any minimal partial overlay of K on G that includes u. Let H be the subgraph of G covered by ψ, and let $P \subseteq V_H$ be the set of nodes on the path from the root of G to u in H. Let ϕ be any minimal partial overlay from K_x to G_u. Then $\psi[P] \cup \phi$ is a minimal partial overlay from K to G, and for every node v in G_u, $image_{K_x}(v) = image_K(v)$. Since K is partially compact for G, K_x is partially compact for G_u.

We show that K_x is partially perfect for G_u by induction on the distance of u from a sink. The assertion clearly holds for the sinks of G. Suppose the assertion is true for nodes at distance less than d from a sink, and let $u \in V_G$ be at distance d from a sink. Let $image_K(u) = x$, let $y_1, \ldots, y_m$ be the children of x in K, and let $v_1, \ldots, v_n$ be the children of u in G. Since K is a PPCS, for each edge $e_i = (u, v_i)$, there is a partial overlay ϕ_i that includes e; since $\phi_i(u) = x$, $1 \leqslant i \leqslant n$, it must be the case that $\phi_i(v_i) = y_j$, for some j, $1 \leqslant j \leqslant m$, and so $image_K(v_i) = y_j$; by the induction hypothesis K_{y_j} is partially perfect for G_{v_i}. Let e be an edge in G_u. We consider two cases to show that for every edge e in G_u, there is a minimal partial overlay of K_x that includes e:

Case 1: $e = (u, v_i)$, $1 \leqslant i \leqslant n$. Since K is a PPCS for G, there is a minimal partial overlay ϕ of K on G that includes e, with $\phi(u) = x$. The restriction of ϕ to the nodes in G_u is a minimal partial overlay of K_x on G_u that includes e.

Case 2: e is in G_{v_i}, $1 \leqslant i \leqslant n$. Without loss of generality, let $image_K(v_i) = y_1$. Since K_{y_1} is partially perfect for G_{v_i}, there is a minimal partial overlay ϕ_1 of K_{y_1} on G_{v_i} that includes e. Let ϕ' be the partial overlay defined by $\phi'(u) = x$, and let $\phi = \phi' \cup \phi_1$. Then ϕ is a minimal partial overlay of K_x on G_u that includes e.

Thus K_x is partially perfect for G_u. Since K_x is partially perfect for G_u and partially compact for G_u, K_x is a PPCS for G_u. $\square$

Theorem 3.1. *For any relation R, possibly containing null values, and data graph G:*

- *If $R \xrightarrow{K} G$, then K is a PPCS for G.*
- *Conversely, if K is a PPCS for G, then there is a relation R such that $R \xrightarrow{K} G$.*
- *If K is a PPCS for G, and $u \in V_G$ such that $image_K(u) = x$, then K_x is a PPCS for G_u.*

3.1. Minimal and maximal skeletons

A data graph can have more than one PPCS; Fig. 4 shows a data graph and two partially perfect compact skeletons corresponding to it. One PPCS induces a relation with no nulls while the other induces a relation containing null values. There are two interesting cases:

- A PPCS K is *minimal* for data graph G if for every other PPCS K' of G, $R(G, K')$ subsumes $R(G, K)$.
- A PPCS K is *maximal* for data graph G if for every other PPCS K' of G, $R(G, K)$ subsumes $R(G, K')$.

Anticipating Theorem 3.2, the maximal and minimal PPCS for a given data graph G, if they exist, are unique. For the data graph in Fig. 4(a), the skeleton in Fig. 4(b) is the unique minimal

PPCS and the skeleton in Fig. 4(c) is the unique maximal PPCS. The minimal PPCS is "conservative" in the sense that its relation contains just those tuples that are in the relations for every PPCS for G, while the maximal PPCS is "aggressive" in the sense that it contains every tuple in the relation for every PPCS for G. Thus the minimal and maximal PPCS give us upper and lower bounds on the relation corresponding to G.

3.2. PPCS algorithms

Let G be a data graph. Let p_1 and p_2 be paths in G that are isomorphic in the graph-theoretic sense. We say that p_1 and p_2 are *attribute isomorphic* if they satisfy the following property: for any node u in p_1, let v be the corresponding node in p_2; then $attr(u) = attr(v)$. A DAG data graph G is *regular* if it satisfies the following conditions:

- Let p_1 and p_2 be paths from the root of G to the same node u. Then p_1 and p_2 are attribute isomorphic.
- Let $u, v \in V_G$ such that $attr(u) = attr(v) \neq \perp$, and let p_1 and p_2 be paths from the root of G to u and v, respectively. Then p_1 and p_2 are attribute isomorphic.
- Let u and v be nodes at the same distance from the root of G. If $attrset(u) \cap attrset(v) \neq \emptyset$, then $attr(u) = attr(v)$.

An interesting characterization of partially perfect compact skeletons is that a data graph G has a PPCS if and only G is regular (Theorem 3.2). The following lemma proves one side of the characterization. We next present an algorithm to compute the maximal partially perfect compact skeleton for a regular data graph G, thus completing the characterization. The algorithm to compute the minimal partially perfect compact skeleton is similar in flavor.

Lemma 3.3. *If a data graph G is not regular, then G has no PPCS.*

Proof. Suppose K is a PPCS for G. The following cases correspond to those in the definition of regularity:

- Let p be some path from the root of G to u. Let $v = image_K(u)$ and let q be the unique path from the root of K to v. Since K is a PPCS for G, there is a minimal partial overlay from K to G that includes p. Since ϕ maps the root of G to the root of K and u to v, it follows that ϕ maps the path p to the path q (since q is the unique path from the root of K to v). Thus p is attribute isomorphic to q. Thus paths p_1 and p_2 are attribute isomorphic to q and therefore to each other.
- Let x be the unique node in K labeled with $attr(u)$ (or $attr(v)$), and let q be the unique path from the root G to x. Then as in the previous case, p_1 and p_2 are attribute isomorphic to q and therefore to each other.
- Suppose u and v are such that they are the same distance d from the root of G and let $A \in attrset(u) \cap attrset(v)$. In K there is a unique path from the root of K to the node labeled A, and there is a unique node x (if any) on the path and at distance d from the root. It follows that $image_K(u) = image_K(v) = x$ and therefore $attr(u) = \perp$ or $attr(u) = attr(x)$, and $attr(v) = \perp$ or $attr(v) = attr(x)$. □

Algorithm. ComputeMaximalPPCS

```
1. If $G$ is not regular, then $G$ has no PPCS.
2. Process nodes of $G$ in decreasing order of their distances from the root. Let $S_d$ be the set of
   nodes at distance $d$ from the root.
3. Process node $u \in S_d$ as follows:
    (i) If $u$ is a leaf, set $T_u$ to be the tree with a single node with label $attr(u)$.
    (ii) Otherwise let $T_1, \ldots, T_m$ be the set of distinct trees corresponding to the children of $u$. Set
         $T_u$ to be the tree with root labeled $attr(u)$ and subtrees $T_1, \ldots, T_m$.
4. For each pair of nodes $u, v \in S_d$ with $attrset(u) \cap attrset(v) \neq \emptyset$, do the following:
    (i) If $attr(u) \neq \perp$, set $A = attr(u)$. Otherwise set $A = attr(v)$.
    (ii) Construct $T$ as follows: the root of $T$ is a node labeled $A$; each distinct subtree of the root
         of either $T_u$ or $T_v$ is a child of the root of $T$.
    (iii) Set $T_u := T$ and $T_v := T$.
5. Let $r$ be the root of $T$. $T_r$ is a maximal PPCS for $G$.
```

Example 3.1. Fig. 5 shows a data graph and the partial skeletons constructed at different stages of processing. The figure labeled (2,3) is obtained after running Step 4 on the skeletons labeled (2) and (3). The final skeleton, corresponding to the root of the data graph, is a maximal PPCS for the website, and the corresponding relation is $\{ab\perp, \perp bc\}$.

Lemma 3.4. *Let G be a regular DAG data graph. When Algorithm ComputeMaximalPPCS terminates, T_r is the unique maximal PPCS for G.*

Proof. By induction on the order in which nodes are processed, we prove the following assertions after node u is processed:

1. T_u is a PPCS of G_u
2. if S is any PPCS of G_u and ϕ is any partial overlay of S on G_u that covers a subgraph H of G_u, there is partial overlay ϕ' of T_u on G_u that covers H.

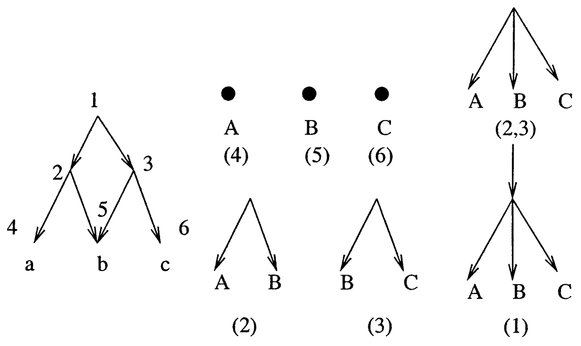


Fig. 5. Algorithm ComputeMaximalPPCS.

Clearly, the assertions are true after nodes at the maximum distance from the root are processed (since these must all be leaf nodes containing non-null values). Suppose the assertions hold after processing all nodes at distance less than d. Let $u \in S_d$ in Step 2.

After Step 3, we claim that T_u is a PPCS for G_u. Let $v_1, \dots, v_n$ be the children of u, and let x_i be the root of the subtree corresponding to T_{v_i} in T_u. Let e be any edge in G_u. If $e = (u, v_i)$ for some $1 \leqslant i \leqslant n$, the partial overlay from the edge (u, x_i) includes e. Else e is in G_{v_i}, $1 \leqslant i \leqslant n$. Since T_{v_i} is a PPCS for G_{v_i} (by induction hypothesis), there is a subgraph S of T_{v_i} that covers e in G_{v_i}. Add the edge (u, x_i) to S to construct a subgraph S' of T_u; clearly S' covers e in G_u. Thus T_u is partially perfect for G. To see that T_u is also partially compact, note that each T_{v_i} is partially compact for G_{v_i}, and any partial overlay of T_u maps the root of T_u to u and nodes in T_{v_i} to nodes in G_{v_i}, $1 \leqslant i \leqslant n$. Thus T_u is a PPCS for G after Step 3.

Suppose T_u is set to T in Step 4. Note that T_u is a subgraph of T with the same root, and so T is also perfect for G_u. Moreover, the subtrees of the root of T have disjoint attribute sets (otherwise they would have been identified in Step 4(c)), so that the only nodes from T_u can participate in any minimal partial overlay from T to G_u, and so T is also partially perfect for G_u. Thus T is a partially perfect compact skeleton for G_u after Step 4; since T_u is set to T, T_u is a PPCS for G_u after Step 4.

To show that assertion (2) holds after Step 3, let S be any PPCS of G_u and consider some partial overlay ϕ of S that covers subgraph H of G_u. Let $x_1, \dots, x_l$ be the children of the root s of S. From Lemma 3.2, each S_{x_i}, $1 \leqslant i \leqslant l$, is a PPCS for some G_{v_i}, $1 \leqslant i \leqslant n$. Thus ϕ can be decomposed as follows: $\phi(s) = u$, and the rest of ϕ agrees with some collection of partial overlays of partially perfect compact skeletons for G_{v_i}, $1 \leqslant i \leqslant n$. By induction hypothesis, ϕ agrees with some set of partial overlays of T_{v_i}, $1 \leqslant i \leqslant n$. Thus, we can construct a partial overlay ϕ' of T_u that covers the same subgraph H. Thus assertion (2) holds after Step 3. If T_u is set to T in Step 4, note that T_u is a subgraph of T with the same root, and so any partial overlay of T_u is also a partial overlay of T. Thus assertion (2) also holds after Step 4.

Assertions 1 and 2 follow by induction and imply the lemma. □

Theorem 3.2. *For any relation R, possibly containing null values, data graph G, and skeleton K:*

- *G has a PPCS if and only if G is regular.*
- *If G has a PPCS, then it has a unique maximal PPCS and a unique minimal PPCS.*
- *There is a polynomial-time algorithm that determines whether G has a PPCS and, if so, computes the minimal and maximal PPCS for G.*

4. Answering queries using compact skeletons

Suppose data graph G has PCS or PPCS K. We now show how to answer queries over the data graph (and the underlying website). We start with the important special case of materializing the entire relation. For simplicity we present our algorithm in the context of perfect compact skeletons; it is straightforward to extend it to partially perfect compact skeletons and also to best-fit skeletons (to be described in Section 6).

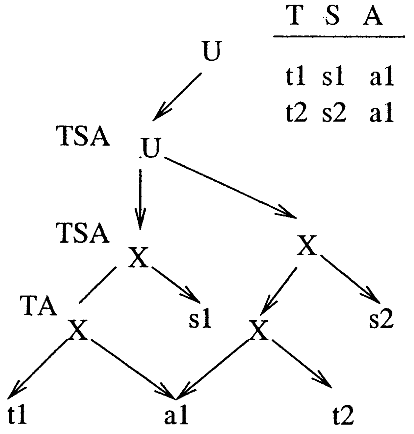
Algorithm. ComputeRelation

```
1. For every node $u$ of $G$ such that $attr(u) \neq \perp$, $R'_u$ is the relation over the single attribute $attr(u)$
   that contains the single tuple $val(u)$.
2. For every sink $u$ of $G$, $R_u := R'_u$.
3. Process the nodes of $G$ bottom up by successive elimination of sinks. Suppose we are currently
   processing node $u$.
   (i) Let $v = image_K(u)$, and let $v_1, \dots, v_n$ be the children of $v$ in $K$. Process each $v_i$ in sequence,
       for $1 \leqslant i \leqslant n$:
       • Let $C_i = \{x \mid (u,x) \in E_G \text{ and } image_K(x) = v_i\}$
       • Set $S_i := \bigcup_{x \in C_i} R_x$.
   (ii) Set $R_u := S_1 \times \cdots \times S_n$.
   (iii) If $attr(u) \neq \perp$, set $R_u := R'_u \times R_u$.
4. If $s$ is the root of $G$, then $R_s$ is the desired result.
```

A useful way to visualize Algorithm ComputeRelation is as a transformation that converts a data graph G into a relational operator DAG. Fig. 6 shows the operator DAG and result relation obtained by running Algorithm ComputeRelation on a portion of the data graph in Fig. 2(a) using the compact skeleton shown in Fig. 2(b).

Theorem 4.1. *Algorithm ComputeRelation computes the relation R corresponding to a data graph G and runs in time $O(k|V_G||R|\log|R|)$ where k is the number of attributes in the schema.*

Proof. The proof of correctness is by induction on the number of iterations of the outer loop of the algorithm; after node u is processed, we show that R_u is the relation corresponding to G_u. For the basis, note that R_u computed for the sinks of G in Step 2 is the relation at those nodes. Suppose the assertion is true for all the children of node u, and we are currently processing node u in Step 3. Let $u_1, \dots, u_m$ be the children of u.



T	S	A
t1	s1	a1
t2	s2	a1

Fig. 6. Running Algorithm ComputeRelation.

Let $t \in R(G_u, K_v)$. There is an overlay ϕ of K_v on G_u that produces t. Let $X_i = attrset(v_i)$, $1 \leqslant i \leqslant n$. From the proof of Theorem 2.2, $X_i \cap X_j = \phi$ and $attr(u) \notin X_i$, $1 \leqslant i \leqslant n$, so the cross products in Steps 3(b) and (c) are well-defined. Define ϕ_i, $1 \leqslant i \leqslant n$ to be the restriction on ϕ to the nodes in K_{v_i}. Then each ϕ_i corresponds to an overlay of K_{v_i} on some G_{u_j} such that $image_K(u_j) = v_i$. By induction hypothesis, there is a tuple $t_i = t[X_i] \in R_{u_j}$ corresponding to overlay ϕ_i, and so $t_i \in S_i$ after Step 3(a). If $attr(u) = \perp$, the tuples t_i, $1 \leqslant \imath \leqslant n$, will combine in the cross product at Step 3(b) to yield the tuple t. If $attr(u) \neq \perp$, then t_i, $1 \leqslant i \leqslant n$ and the single tuple in R'_u will combine in the cross product at Step 3(c) to yield the tuple t. Thus $t \in R_u$.

Conversely, let $t \in R_u$. Then for each i, $1 \leqslant i \leqslant n$, $t_i = t[X_i] \in R_x$ for some $x \in C_i$. Each t_i corresponds to an overlay ϕ_i of K_{v_i} on G_x. Define ϕ as follows: $\phi(u) = v$ and $\phi = \phi_i$ on the nodes in K_{v_i}. Then ϕ is an overlay of K_v on G_u that derives tuple t, and so $t \in R(G_u, K_v)$.

We have thus shown that $R_u = R(G_u, K_v)$. Correctness follows by induction. The complexity follows from the observation that each cross product can be done in time proportional to the size of the output, and each union (with duplicate elimination) can be done in time $O(k|R|\log|R|)$. □

To extend Algorithm *ComputeRelation* to partially perfect compact skeletons, we need only modify Step 3(a) as follows. Whenever the set C_i is empty, we set S_i to be the relation over the set of attributes in $attrset(v_i)$ that has a single tuple with null values for all columns. The proof of correctness is extremely similar to that of Theorem 2.5 and the running time remains unchanged. In an earlier section, we had asked whether we can characterize those data graphs G for which there is a unique relation R such that $R \xrightarrow{K} G$. We now provide such a characterization.

Corollary 4.1. *Given data graph G with PCS K, there is a unique relation R such that* $R \xrightarrow{K} G$ *if and only if whenever Algorithm ComputeRelation is required to compute a cross product, at most one relation in the cross product is non-singleton.*

Proof. The relation associated with G is not unique iff there is more than one set of overlays of K that include every edge of G such that each set of overlays induces a different relation. It is a straightforward induction that if there is a unique relation for G, there is a unique relation at every node of G.

Suppose u is a node of G such that S_1 and S_2 are non-singleton in Step 3. As in the proof of the preceding theorem, we can construct an overlay ϕ of G_u corresponding to every combination of tuples in $S_1, \dots, S_n$. Suppose, without loss of generality, that $S_1 = \{t_{11}, t_{12}\}$, $S_2 = \{t_{21}, t_{22}\}$, and $S_i = \{t_i\}$, $3 \leqslant i \leqslant n$ (the case where more than two of the sets are non-singleton is similar). Let ϕ_1 be the overlay corresponding to $t_{11}, t_{21}, t_3, \dots, t_n$, ϕ_2 the overlay corresponding to $t_{12}, t_{22}, t_3, \dots, t_n$, ψ_1 the overlay corresponding to $t_{11}, t_{22}, t_3, \dots, t_n$, and ψ_2 the overlay corresponding to $t_{12}, t_{21}, t_3, \dots, t_n$. It can be verified that $\{\phi_1, \phi_2\}$ and $\{\psi_1, \psi_2\}$ are sets of overlays that include every edge of e and induce different relations.

Conversely, suppose the relation associated with G is not unique. Let u be a node of G at the least distance from a sink such that the relation at G_u is not unique. It can be shown that the there is more than one non-singleton relation in the cross product at Step 3 for u. Suppose not; without loss of generality, S_1 is non-singleton and S_i is singleton for $2 \leqslant i \leqslant n$. Constructing overlays as before, there is a unique set of overlays that covers every edge of G; each such overlay corresponds to some tuple from S_1 and the unique tuple in S_i, $2 \leqslant i \leqslant n$. Thus there is a unique relation associated with G_u, a contradiction. $\square$

4.1. Answering queries

Often in a mediation scenario, the query to a wrapper does not materialize the entire relation but some subset of it. We could answer such queries by first computing the relation corresponding to the website, and then evaluating the query on the materialized relation. However, it may be the case that the entire relation is large while the result of the query is relatively small; thus, it would be helpful to have output-size sensitive algorithms to answer arbitrary queries on a data graph.

Suppose G is a data graph with PCS K, corresponding to relation R over attribute set X. We now consider query plans for select-project queries on R, that is, queries of the form $\pi_Y(\sigma_{A=a}R)$, where $Y \subseteq X$ and A is in X. The most expensive operation in such cases is fetching web pages, so we look for query plans that fetch as few pages as possible. In the interest of conciseness, we sketch here the intuition behind the query-answering algorithm, and omit a full listing of the algorithm.

A select-project query on R can be seen as a transformation on the PCS K. Consider the project query $\pi_Y(R)$. Let $K[Y]$ be the unique connected subgraph of K that includes the root of K and a path from the root to every node labeled with some A in Y. Let Y' be the attribute set of $K[Y]$; clearly $Y \subseteq Y'$ and $K[Y] = K[Y']$. It can be verified that $\pi_{Y'}(R)$ is the relation corresponding to subgraph of G that has $K[Y']$ as its PCS. To construct this relation, we can avoid fetching pages that do not contain an information element from Y'. We then project out attributes in $Y' - Y$ to obtain $\pi_Y(R)$.

The query $\sigma_{A=a}(R)$ can be seen as adding the constraint $A = a$ to the node in K labeled by attribute A; call the resulting constrained skeleton K'. We extend the definition of overlays to skeletons with constraints in the natural manner. Now the answer to the query $\sigma_{A=a}(R)$ is the relation corresponding to the constrained skeleton K'. We may modify Algorithm ComputeRelation to first fetch pages containing values of attribute A and avoid fetching pages corresponding to cross product operands when we can determine that one of the operands of the cross product is the empty relation. An equivalent view is that we push the selection condition $A = a$ down the operator DAG induced on G by K.

To answer a general select-project query, we combine the techniques for selections and projections.

5. Noisy data graphs

In addition to incomplete information, real websites contain *noise*, i.e., superfluous information. Such a noisy website may not have a PPCS. Noise in websites can be purely random, but can also result from false-positive matches from the patterns used to identify data elements. For example, a pattern to identify US states might match the "MS" in the text "MS Word," and a pattern to identify salaries like "70K" might match the text "401K," which in most cases refers to a retirement plan rather than a salary. In addition, there are sometimes links in websites that do not correspond to skeleton links. For example, consider a website for an online retailer, with products organized by category, then by subcategory. There might be non-skeleton links directly from the home page to some of the product pages because these are "featured products" on a given day.

When confronted with a noisy website, we look for a skeleton that covers as much of the data graph as possible. In doing so we must relax both the compactness condition and the perfect coverage condition for skeletons. We relax the compactness condition as follows. Given a data graph G and a skeleton K, a set S of overlays is *consistent* if for every node u in G that is included in some overlay in S, there is a unique node v in K such that every overlay $\phi \in S$ that includes u maps u to v. The *cover* of set S is the set of nodes and edges of G that are included in some overlay $\phi \in S$, and the *coverage* of S is some metric that measures the goodness of the cover. Possible coverage metrics include the cardinality (i.e., the number of nodes and edges) of the cover, the number of nodes in the cover, and the number of non-null data values in the cover. In the rest of this chapter, we define the coverage as the number of nodes in the cover; our results, however, extend to the other metrics as well.

The *coverage of skeleton K*, denoted $coverage(G, K)$, is the maximum coverage across all consistent sets of overlays. A skeleton K is a *best-fit skeleton* for a data graph G if for any other skeleton K', $coverage(G, K) \geqslant coverage(G, K')$.

Unfortunately, the problem of finding a best-fit skeleton for a given data graph is NP-complete, even when the data graph is restricted to be a tree of depth no greater than 3. We formally define the *BFS decision problem* as follows: Given a data graph G and a constant c, determine whether there is a skeleton K such that $coverage(G, K) \geqslant c$. We show that the BFS decision problem is NP-complete through a reduction from the *set cover* problem, which is known to be NP-complete [16]. The set cover problem is defined as follows: Given a collection of sets $\mathscr{S} = S_1, \ldots, S_m$, and a constant $k \leqslant m$, determine whether there is a collection of k sets, $S_{i_1}, \ldots, S_{i_k}$, such that $\bigcup_{1 \leqslant j \leqslant k} S_{i_j} = \bigcup_{1 \leqslant i \leqslant m} S_i$. We assume that $\bigcup_{1 \leqslant i \leqslant m} S_i = \{A_1, \ldots, A_n\}$.

Theorem 5.1. *The BFS decision problem is NP-complete for tree data graphs of depth* 3.

Proof. To verify membership in NP, note that we can guess nondeterministically a skeleton K, a set S of k nodes in G, and partial overlays that cover every node in S (we need to guess at most k consistent partial overlays). To prove NP-hardness, we show that the set cover problem can be reduced to the BFS problem. Given an instance $\mathscr{S}$ of the set cover problem, construct a tree data graph G as follows (Fig. 7 suggests the construction). The schema has the attributes $S_1, \ldots, S_m, A_1, \ldots, A_n$, and additional attributes B, $C_1, \ldots, C_n$, and $D_1, \ldots, D_m$. The root r of G has $attr(r) = \perp$, and has $m + 1$ children $u_1, \ldots, u_{m+1}$, with $attr(u_i) = S_i$, $1 \leqslant i \leqslant m$ and $attr(u_{m+1}) = B$. Whenever $A_j \in S_i$, u_i has a child v_{ji} with $attr(v_{ji}) = A_j$. Each node v_{ji} has $2m + 1$ children, each with a data value corresponding to attribute C_i. The node u_{m+1} has m children, $w_1, \ldots, w_m$, with $attr(w_i) = S_i$, $1 \leqslant i \leqslant m$. Each node w_i, $1 \leqslant i \leqslant m$, has one child with a data value corresponding to attribute D_i.

Observe that for each value of i, the nodes v_{ij} all have the same attribute A_i, while the parent of v_{ij} has attribute S_j. Therefore in any set of consistent overlays, at most one of the nodes v_{ij} for

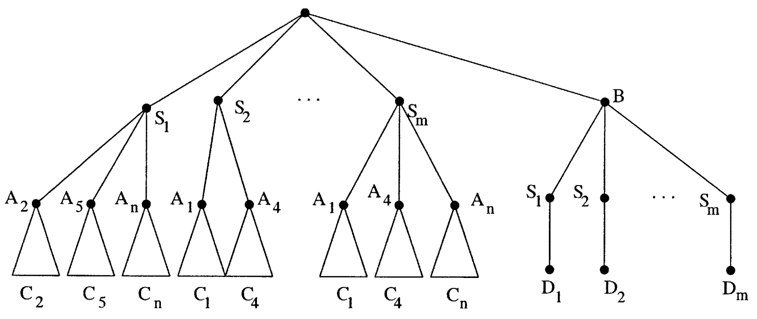


Fig. 7. Data graph G for proof of Theorem 5.1.

some value of j can be in the cover for each i, $1 \leqslant i \leqslant n$. Counting the nodes in G with this restriction in mind, the maximum number of nodes in any cover of G is $N = 2mn + 3m + 2n + 2$. We claim that there is a skeleton K with $coverage(G, K) \geqslant N - m - k$ if and only if $\mathcal{S}$ has a set cover of size k.

IF: Suppose $\mathcal{S}$ has a set cover of size k. Assume without loss of generality that the set cover is $S_1, \ldots, S_k$. Construct skeleton K as follows: The root of K has no data value, and has $k+1$ children $x_1, \ldots, x_{k+1}$, with $attr(x_i) = S_i$, $1 \leqslant i \leqslant k$ and $attr(x_{k+1}) = B$. There are nodes y_i, $1 \leqslant i \leqslant n$, $attr(y_i) = A_i$, and nodes $z_{k+1}, \ldots, z_m$, with $attr(z_i) = S_i$, $k+1 \leqslant i \leqslant m$. Each node y_i has a single child with attribute C_i, while each node z_i has a single child with attribute D_i. Since $S_1, \ldots, S_k$ is a set cover, for each A_i, $1 \leqslant i \leqslant n$, there is at least one S_j, $1 \leqslant j \leqslant k$, with $A_i \in S_j$ (if there is more than one set that contains A_i, pick one arbitrarily). Make x_i the parent of y_i in K. It is clear that for each i, $1 \leqslant i \leqslant n$, there are overlays of K on G that cover at least one node with attribute A_i and its children, while all other nodes with attribute A_i and their children are not covered. All other nodes of G are covered except the nodes $u_{k+1}, \ldots, u_m$, and the nodes $w_1, \ldots, w_k$ and their children. Counting these nodes gives

$$coverage(G, K) = N - (m - k) - 2k = N - m - k.$$

ONLY IF: Suppose G has a skeleton K with $coverage(G, K) \geqslant N - m - k$. Observe that in this case, for each i, $1 \leqslant i \leqslant n$, at least one node with attribute A_i must be in the cover; if not, the maximum possible coverage is $N - (2m + 2)$, which is strictly less than $N - m - k$ since $k \leqslant m$. Observe also that for $1 \leqslant i \leqslant m$, both u_i and w_i cannot be in the cover, since $attr(u_i) = attr(w_i) = S_i$, and u_i and w_i are at different distances from the root of G. We claim that at most k of the nodes among the u_i, $1 \leqslant i \leqslant m$, are in the cover. Suppose not, if $k' > k$ nodes among the u_i, $1 \leqslant i \leqslant m$, are in the cover. Then at most $m - k'$ nodes among the w_i are in the cover, and so

$$coverage(G, K) \leqslant N - (m - k') - 2k' = N - m - k' < N - m - k$$

since $k' > k$. Thus the cover must include exactly one node with attribute A_i, $1 \leqslant i \leqslant n$, and at most k of the nodes among the u_j, $1 \leqslant j \leqslant m$. The sets in $\mathcal{S}$ corresponding to the nodes u_j that are in the cover constitute a set cover for $\mathcal{S}$ of cardinality at most k. $\square$

Given this negative result, there are two approaches to constructing a "good" skeleton for a data graph:

1. *Semi-automatic*: Given a data graph, a human uses a visualization technique to "eyeball" the data graph and guesses a skeleton. The system computes the coverage of the skeleton. If the coverage is large enough as a ratio of the data graph size, the skeleton is accepted as a good skeleton.
2. *Automatic*: Given a data graph, use heuristics to compute a good skeleton that is as close to the best-fit skeleton as possible.

In this section, we explore computing the coverage of a skeleton to aid the semiautomatic approach. Section 6 consider heuristics to compute best-fit skeletons.

5.1. Computing the coverage of a skeleton

Given a data graph G, the best-fit skeleton is not necessarily unique; this observation ollows from our experience with partially perfect compact skeletons, which are a special case of best-fit skeletons with perfect coverage. It turns out that we can restrict our attention to a class of skeletons that we call *canonical skeletons*. A skeleton K is canonical if each labeled node in K has at most one unlabeled child. Canonical skeletons are a natural generalization of minimal partially perfect compact skeletons, and are important because of the following result.

Lemma 5.1. *Given an arbitrary data graph G and skeleton K, there is a canonical skeleton K' such that $coverage(G,K') \geqslant coverage(G,K)$.*

Intuitively, given a non-canonical skeleton we can identify unlabeled siblings to construct a canonical skeleton with the same or better coverage. It can be shown that every partial overlay of the original skeleton is also a partial overlay of the modified skeleton. The following example provides a flavor of the constructive proof.

Example 5.1. Consider the data graph G in Fig. 8(a) and the skeleton K in Fig. 8(b). It can be verified that $coverage(G,K) = 6$. The canonical skeleton K' in Fig. 8(c) is obtained by identifying unlabeled siblings of K. It can be verified that $coverage(G,K') = 7$, and so $coverage(G,K') \geqslant coverage(G,K)$.

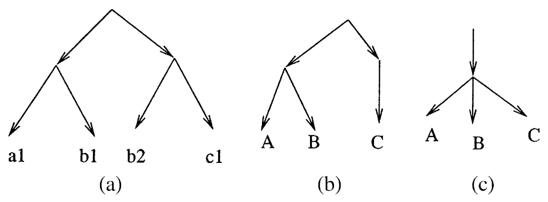


Fig. 8. A data graph and two skeletons.

It turns out that the problem of computing the coverage of a skeleton for a data graph is NP-complete in the general case. We formally define the *skeleton coverage decision problem* as follows: Given a data graph G, a skeleton K, and a constant c, determine whether $coverage(G, K) \geqslant c$.

Theorem 5.2. *The skeleton coverage decision problem is NP-complete.*

Proof. To verify membership in NP, note that we can guess nondeterministically a set S of k nodes in G, and partial overlays that cover every node in S (we need to guess at most k consistent partial overlays).

To show NP-hardness, we prove that the set cover problem can be reduced to the skeleton coverage problem. Consider an instance $\mathscr{S} = S_1, \ldots, S_m$ of the set cover problem, and let $\bigcup_{1 \leqslant i \leqslant m} S_i = \{A_1, \ldots, A_n\}$. Construct a data graph G and a skeleton K as follows. The schema has attributes $S_1, \ldots, S_m, B_1, \ldots, B_m, C_1, \ldots, C_m$, and an attribute A_{ij} whenever $A_i \in S_j$, $1 \leqslant i \leqslant n$, $1 \leqslant j \leqslant m$. The root of data graph G has no data value, and has children $u_1, \ldots, u_m$, with $attr(u_i) = S_i$, $1 \leqslant i \leqslant m$. Each node u_i has one child v_i with $attr(v_i) = \perp$. Each node v_i has a child w_i with $attr(w_i) = B_i$. Each node w_i has a single child with data value corresponding to attribute C_i. There is a node x_i corresponding to each A_i, $1 \leqslant i \leqslant n$, with $attr(x_i) = \perp$. The arc (v_j, x_i) is in G whenever $A_i \in S_j$. Whenever $A_i \in S_j$, the node x_i has $2m + 1$ children with data values corresponding to attribute A_{ij}. Fig. 9(a) suggests the data graph G.

Construct skeleton K as follows. The root of K has no data value and has m children, one each with attribute S_i, $1 \leqslant i \leqslant m$. Each child labeled S_i has two children, y_i and z_i, with null attributes. The child y_i has a single child with attribute B_i, which in turn has a single child with attribute C_i. The child z_i has a single child z'_i with a null attribute, and z'_i has a child A_{ij} whenever $A_i \in S_j$. Fig. 9(b) suggests the skeleton K.

Observe that in any set of consistent overlays that covers node x_i in G, x_i is mapped to some node z'_j in K. Therefore the corresponding cover includes all the nodes with data values corresponding to attribute A_{ij}, but none of the nodes with data values corresponding to attributes A_{il} for $l \neq j$. Therefore, any cover includes at most $2m + 1$ children of each node x_i, $1 \leqslant i \leqslant n$. Counting the nodes of G with this restriction in mind, we determine that $coverage(G, K) \leqslant 2mn + 4m + 2n + 1$. Let us define $N = 2mn + 4m + 2n + 1$. We claim that $coverage(G, K) \geqslant N - 2k$ if and only if $\mathscr{S}$ has a set cover of size k.

If: Suppose $\mathscr{S}$ has a set cover of size k. Assume, without loss of generality, that the set cover is $S_1, \ldots, S_k$. Consider the set of overlays that maps each v_i, $1 \leqslant i \leqslant k$, to the corresponding node z_i in K, and each node v_i, $k + 1 \leqslant i \leqslant m$, to node y_i in K. Extend this set of overlays into a cover. Since $S_1, \ldots, S_k$ is a set cover for $\mathscr{S}$, each node x_i, $1 \leqslant i \leqslant n$, can be mapped to some z'_{ij} for $1 \leqslant j \leqslant k$, and so the cover includes some $2m + 1$ children of x_i with attributes A_{ij}. A simple count shows that the number of nodes in the cover is $N - 2k$.

Only if: Suppose $coverage(G, K) \geqslant N - 2k$. The maximal cover must include each node x_i, $1 \leqslant i \leqslant n$; if not, the size of the cover would be at most $N - (2m + 2) < N - 2k$ since $k \leqslant m$. An argument similar to that used in the corresponding case in the proof of Theorem 5.1 shows that $\mathscr{S}$ has a set cover of size k. $\square$

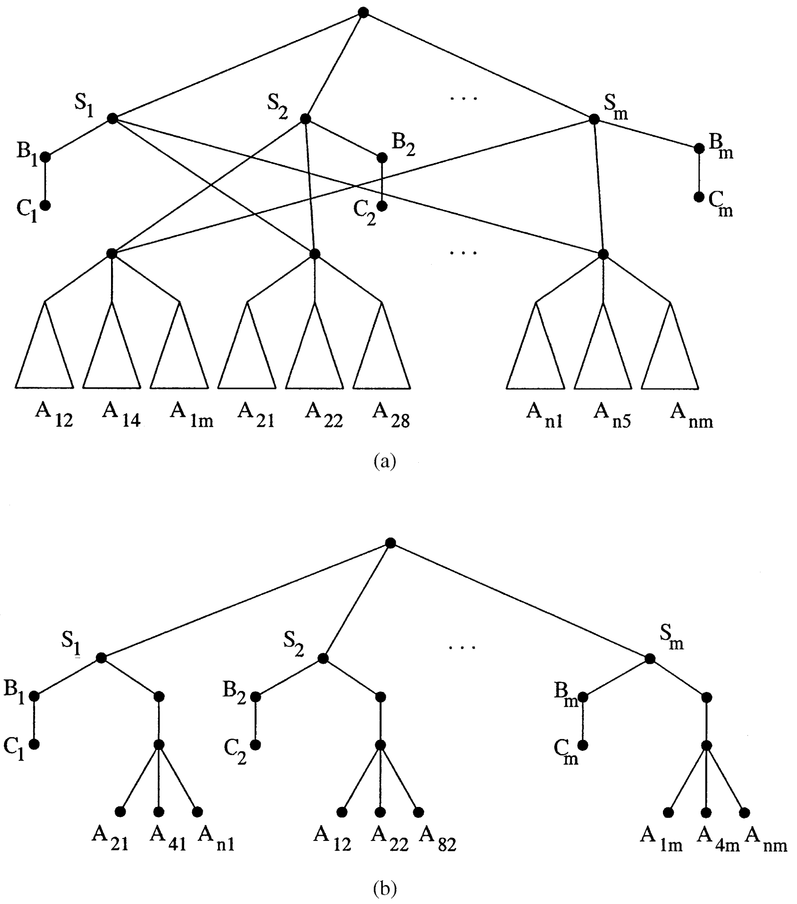


Fig. 9. Data graph G and skeleton K for proof of Theorem 5.2.

It is natural to ask whether we can impose some practical restrictions on the structure of the data graph and the skeleton to yield tractable cases. We consider two such tractable cases:

1. Data graphs that are trees. In this case, there is a simple linear algorithm to compute the coverage.

2. A class of DAG data graphs called *unambiguous data graphs* that we define in Section 5.3. Informally, a data graph G is unambiguous if we can partition its null-valued nodes into equivalence classes, such that every null-valued node has all its parents in the same equivalence class. In this case, there is a polynomial-time algorithm to compute the coverage of a canonical skeleton. This case is interesting because in practice many data graphs corresponding to real websites are unambiguous.

5.2. Tree data graphs

Given a tree data graph G and a skeleton K over the same schema, there is a simple bottom-up algorithm to compute $coverage(G, K)$.

Algorithm. TreeCoverage

```
1. Process each leaf node $u$ of $G$ as follows:
    (i) Let $x$ be the node in $K$ with $attr(x) = attr(u)$.
    (ii) Set $coverage(u, x) := 1$.
    (iii) Set $coverage(u, y) := 0$ for all nodes $y \in V_K$ with $y \neq x$.
2. Process nodes of $G$ bottom-up, and let $u$ be a node all of whose children have been processed.
   Process $u$ as follows:
    (i) Let $P = \{x \in V_K \mid attr(x) = attr(u)\}$
    (ii) Set $coverage(u, x) := 1$ for each $x \in P$ and $coverage(u, y) := 0$ for $y \in V_K$ with $y \notin P$.
    (iii) For each $x \in P$, update $coverage(u, x)$ as follows:
        a. Let $C = children(x)$
        b. For each child $v$ of $u$ set

           $coverage(u, x) := coverage(u, x) + \max_{y \in C} coverage(v, y)$

3. $coverage(G, K) = coverage(r_G, r_K)$, where $r_G$ is the root of $G$ and $r_K$ is the root of $K$.
```

Example 5.2. Consider the skeleton K and data graph G shown in Fig. 10. We have used numbers $1, 2, \ldots$ to identify nodes in K and G. In Algorithm TreeCoverage, we compute the following non-zero coverages:

- For the leaf nodes, $coverage(4, 4) = coverage(5, 5) = coverage(6, 5) = coverage(7, 6) = 1$
- For node 2, $coverage(2, 2) = 3$, $coverage(2, 3) = 1$, and $coverage(2, 1) = 1$.
- For node 3, $coverage(3, 2) = 2$, $coverage(3, 3) = 2$, and $coverage(3, 1) = 1$.
- For the root, $coverage(1, 1) = 6$, $coverage(1, 2) = 1$, and $coverage(1, 3) = 1$.

Therefore we have $coverage(G, K) = coverage(1, 1) = 6$.

Theorem 5.3. *Given a tree data graph G and a skeleton K, Algorithm TreeCoverage computes coverage (G, K) and runs in time $O(|V_K||V_G|)$.*

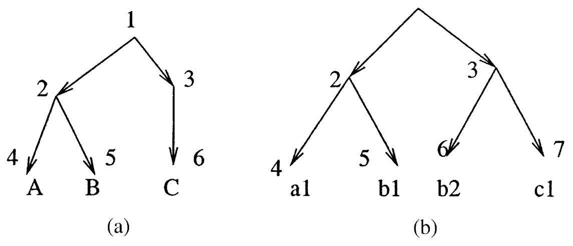


Fig. 10. Algorithm TreeCoverage.

Proof. We use induction on the number of nodes of G that have been processed by the algorithm, and show that after node u has been processed, for any node x in K, $coverage(u,x) = coverage(G_u, K_x)$. For the basis, note that after Step 1, the assertion holds for the leaf nodes of G.

Suppose that the assertion holds for the children of node u and that we have processed node u. Let $x \in V_K$. We consider two cases:

Case 1: If $x \notin P$ after Step 2(i), it is easy to see that $coverage(G_u, K_x) = 0$, since $attr(u) \neq attr(x)$ and so u cannot be mapped to x by any partial overlay. Since we set $coverage(u,x) = 0$ for $x \notin P$ in Step 2(ii), in such cases we have $coverage(u,x) = coverage(G_u, K_x)$.

Case 2: Suppose $x \in P$ after Step 2(i), and consider any partial overlay ϕ of K_x on G_u. ϕ maps u to x, and if ϕ maps any child v of u it must map it to some child y of x. Moreover, the mapping for each child v of u is independent of the mapping for any other child (since G is a tree and the children do not share any descendants). Thus for each child v of u, we can choose to map v to some child y of x in such way as to maximize the $coverage(G_v, K_y)$. Finally, the coverages of the children of u are additive since the children do not share any descendants, and so Step 2(iii) computes $coverage(G_u, K_x)$.

Correctness follows by induction. For the complexity, we note that the algorithm processes each node of G once and does work proportional to the number of nodes in K. □

5.3. Unambiguous data graphs

Given a data graph G and a skeleton K, suppose the root of either has a null attribute; we can invent a new attribute and associate it with the roots of G and K, and not affect the value of $coverage(G, K)$ or any of the results in this chapter. Therefore, in what follows we assume without loss of generality that data graphs and skeletons have non-null attributes at their roots.

Informally, a data graph G is unambiguous if we can partition its null-valued nodes into equivalence classes, such that every null-valued node has all its parents in the same equivalence class. We formalize this notion using a process called *labeling* that we now define.

Given a skeleton K, we associate a label with each node of K as follows. We assume without loss of generality that the root r of K has $attr(K) \neq \perp$. For each node $x \in V_K$, the *label* function is defined as follows:

- If $attr(x) = A \neq \perp$, for some attribute A in the schema, $label(x) = \{A_0\}$.
- If $attr(x) = \perp$, let y be the parent of x in K, with $label(y) = A_i$ for some attribute A and $i \geqslant 0$. Then $attr(x) = A_{i+1}$.

Note that if K is canonical, then each node of K has a unique label; if K is not canonical, then each pair of siblings in K with null attributes share the same label.

We now extend the label function to DAG data graphs. The difference is that in a data graph, a node may have more than one parent, and so we must allow for a set of labels associated with each node. Let G be a data graph, and we assume without loss of generality that the root of G has a non-null data value. The *label* function recursively associates a set of labels with each node u in G as follows:

- If $attr(u) = A \neq \perp$, $label(u) = \{A_0\}$.
- If $attr(u) = \perp$, let P be the set of parents of u in G. Then for each node $v \in P$, if $A_i \in label(v)$, then A_{i+1} is in $label(u)$.

A data graph G is *unambiguous* if for each node u in G, $label(u)$ is a singleton set. As a special case, a data graph G is unambiguous if all its nodes have non-null values. In addition, every tree data graph is an unambiguous data graph. In practice, it turns out that many data graphs corresponding to websites, especially those that have database backends, are unambiguous. Since unambiguous data graphs have singleton label sets for each node, we drop the set notation and say $label(u) = A$ to mean $label(u) = \{A\}$.

Example 5.3. Fig. 11(a) shows a canonical skeleton K. Fig. 11(b) shows the labels computed for the nodes of K. Fig. 11(c) shows a data graph G, while Fig. 11(d) shows the result of labeling the nodes of G. Since each node of G has a unique label, G is unambiguous.

Lemma 5.2 states an important property of unambiguous data graphs and leads to a simple algorithm to compute the coverage of a canonical skeleton K for an unambiguous data graph G, which we present below.

Lemma 5.2. *Let G be an unambiguous data graph and K a canonical skeleton. Let $u \in V_G$, $x \in V_K$ with $label(u) = label(x)$. Then in any partial overlay of K on G that includes u, u is mapped to x.*

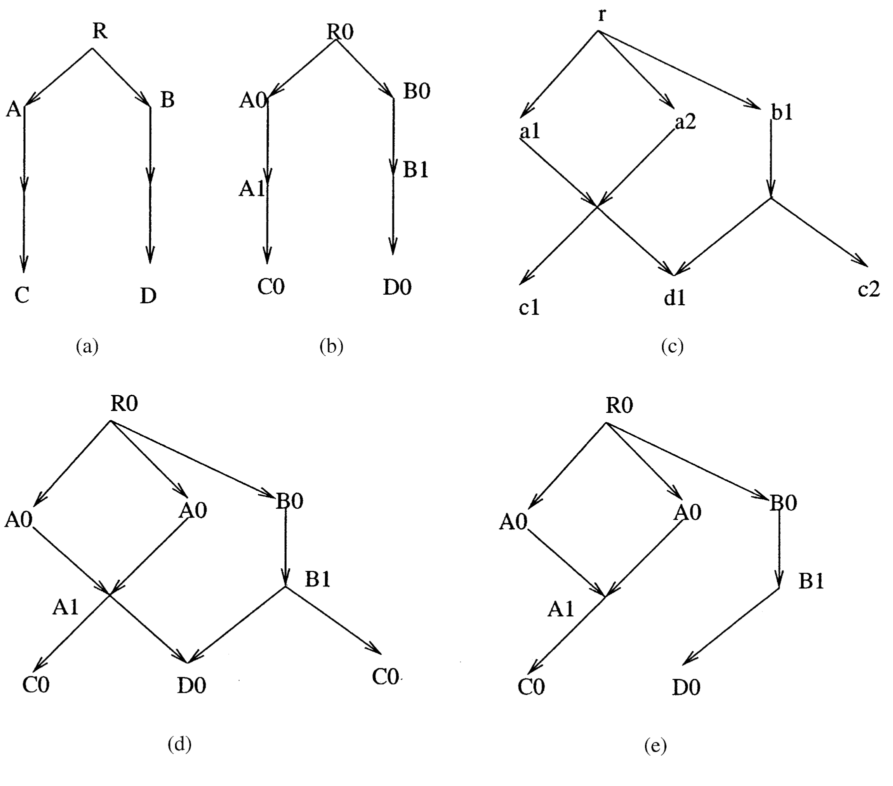


Fig. 11. Algorithm CanonicalCoverage.

Table 1
Complexity of the coverage problem

	Tree data graphs	Unambiguous DAGs	General DAGs
Canonical skeletons	P	P	NP-complete
Arbitrary skeletons	P	NP-complete	NP-complete

Proof. Let ϕ be a partial overlay of K on G that includes u. If $attr(u) \neq \perp$, then $attr(u) = label(u) = label(x) = attr(x)$, and so $\phi(u) = x$. If $attr(u) = \perp$, let $label(u) = label(x) = A_i$, $i > 0$, for some attribute A in the schema. We use induction on i to complete the proof. □

Algorithm. CanonicalCoverage

1. Delete from G all nodes u such that there is no node x in K with $label(x) = label(u)$. Recursively, delete any nodes and edges of G that get disconnected from the root of G.
2. Process the edges of G in any order. Let $e = (u, v)$ be any edge in G.
3. Let y be the node in K with $label(y) = label(v)$, and let $x = parent(y)$.
4. If $label(u) \neq label(x)$, delete edge e from G. Recursively delete any nodes and edges of G that get disconnected from the root of G.
5. $coverage(G, K) = |V_G|$, where $|V_G|$ is the set of nodes that remain in G.

Example 5.4. Let us run Algorithm CanonicalCoverage using the labeled data graph and skeleton from Fig. 11. Fig. 11(e) shows the portion of G that remains at the conclusion of the algorithm. There are 8 nodes remaining in the data graph, and so $coverage(G, K) = 8$.

Theorem 5.4. *Given an unambiguous data graph G and a canonical skeleton K, Algorithm CanonicalCoverage computes $coverage(G, K)$ and runs in time $O(|V_K| + |E_G|)$.*

Proof. Correctness follows from Lemma 5.2. For the complexity, note that we process each edge of G once to compute the label function and once in Algorithm CanonicalCoverage. Similarly, we process each edge of K exactly once to compute the label function for K. □

5.4. Complexity

Table 1 summarizes the complexity of the coverage problem.

6. Computing the best-fit skeleton

We turn now to the BFS problem and ask whether we can restrict the structure of G, as in the case of the coverage problem, to come up with tractable cases. Unfortunately, such is not the case.

From Theorem 5.1, the best-fit problem is NP-complete even when the data graph G is restricted to be a tree of depth 3 with no unlabeled nodes.

We look therefore for polynomial-time heuristics that can approximate a best-fit skeleton. For simplicity, we restrict our attention to unambiguous data graphs. We also make a further simplifying assumption, as follows. Let G be a data graph. The *ancestor* relation among the attributes in the schema is defined in the following manner: attribute A is an ancestor of attribute B if there are nodes $u, v \in V_G$ with $attr(u) = A$, $attr(v) = B$, and u is an ancestor of v in G. The ancestor relation on G is the transitive closure of such pairwise ancestor relationships. If the ancestor relation is acyclic (i.e., it is a partial order), we call G *label-acyclic*. In what follows, we restrict our attention to unambiguous, label-acyclic data graphs. In practice, this assumption is reasonable because many websites of interest have data graphs that fit this model. Our heuristics can be modified for more general data graphs.

Given a data graph G, our algorithms to construct best-fit skeletons work in three steps:

1. Transform G into a *fully-labeled data graph* G' (as defined in Section 6.1).
2. Construct a "good" skeleton K' for G'.
3. Transform K' into a skeleton K for G.

We first describe the label transformation process involved in Steps 1 and 3. We then propose two different heuristics for Step 2 and study how well they perform in practice.

6.1. Labeled data graphs

Let G be an unambiguous data graph over schema X. Label each node in G using the *label* function defined in Section 5.3. Let L be the set of labels. It is useful to think of L as the set of attributes in a new schema. We construct a new data graph $G' = label(G)$ as follows: G' is a copy of G, but its attributes are the set of labels in L. For each node u in G with $label(u) = A_i$, for some $A \in X$ and $i \geqslant 0$, the corresponding node u' in G' has $attr(u') = A$.

We also define a reverse, unlabeling transformation on skeletons. Let K' be a skeleton on schema L. Then $K = unlabel(K')$ is the skeleton over schema X whose node and edge set are copies of K', and whose nodes are labeled as follows:

- For each node u' in K' with $attr(u) = A_0 \in L$, the corresponding node u in K has $attr(u) = A$.
- For each node u' in K' with $attr(u) = A_i \in L$, $i > 0$, the corresponding node u in K is unlabeled.

When searching for a best-fit skeleton, Lemma 5.1 allows us to restrict our attention to canonical skeletons. The following lemma follows from Lemma 5.2 and allows us to work with fully labeled data graphs.

Lemma 6.1. *Let G be an unambiguous data graph, and let $G' = label(G)$. Suppose K' is an canonical skeleton for G' and $K = unlabel(K')$. Then $coverage(G, K) = coverage(G', K')$.*

6.2. The greedy heuristic

Let G be an unambiguous, label-acyclic, fully labeled data graph over schema (label set) L. We now present a simple polynomial-time heuristic that computes a canonical skeleton K for G. Since

K is a tree all of whose nodes are labeled, it can be specified completely by the *parent* function on labels, as follows: suppose u is a node in K with $attr(u) = A_i \in L$, and v is the parent of u in K with $attr(v) = B_j \in L$, then $parent(A_i) = B_j$. As a special case, if R is the label of the root of K, then $parent(R) = \perp$.

Algorithm. GreedySkeleton

```
1. For all pairs of labels $X, Y \in L$, set $parentCount(X, Y) = 0$.
2. Traverse $G$ and process each arc $(u, v)$ as follows: if $attr(u) = X$ and $attr(v) = Y$, set
       $parentCount(X, Y) := parentCount(X, Y) + 1$.
3. Process each label in $X \in L$ as follows:
   (i) Let $Y \in L$ be the label with the largest value of $parentCount(Y, X)$.
   (ii) Set $parent(X) = Y$.
```

Since G is label-acyclic, the *parent* function computed by Algorithm GreedySkeleton is guaranteed to be acyclic. The following example illustrates the algorithm.

Example 6.1. Fig. 12(a) shows a fully labeled data graph G. The non-zero parent count are as follows:

- $parentCount(A_1, C_0) = 1$, and $parent(C_0) = A_1$.
- $parentCount(A_1, D_0) = 1, parentCount(B_1, D_0) = 2$, so $parent(D_0) = B_1$.
- $parentCount(A_0, A_1) = 2$, $parent(A_1) = A_0$.
- $parentCount(B_0, B_1) = 1$, $parent(B_1) = B_0$.
- $parentCount(R_0, A_0) = 2$, $parent(A_0) = R_0$.
- $parentCount(R_0, B_0) = 1$, $parent(B_0) = R_0$.

The resulting skeleton K' is shown in Fig. 12(b). Unlabeling K' yields the skeleton K in Fig. 12(c).

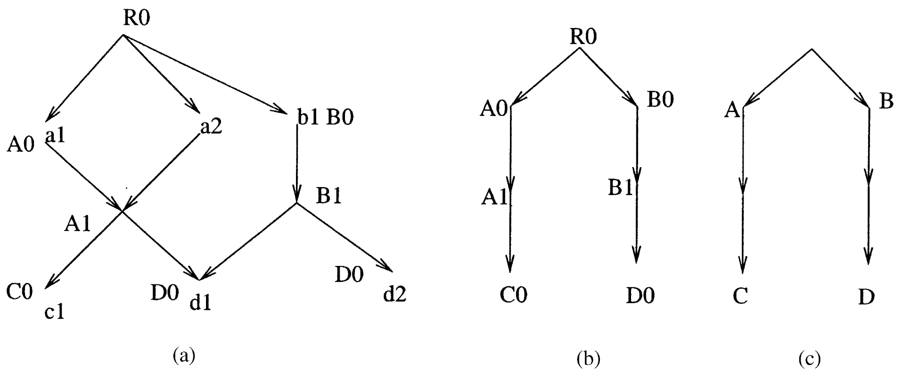


Fig. 12. Running Algorithm GreedySkeleton.

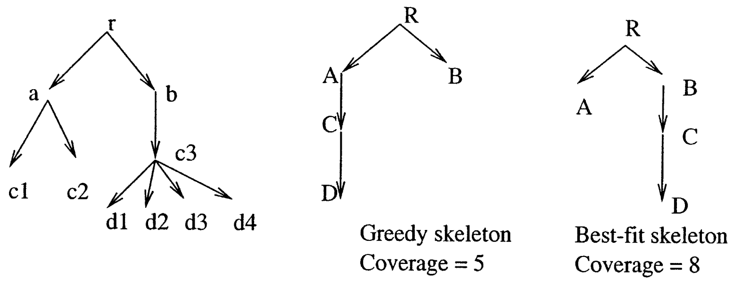


Fig. 13. A data graph where the greedy algorithm does badly.

6.3. The weighted greedy heuristic

Fig. 13 shows an example of a data graph in which the greedy algorithm does not pick the optimal skeleton. Intuitively, the reason why greedy does badly on the data graph in Fig. 13 is that all its decisions are made independently, so that the effects of a prior decision are not factored into subsequent decisions. We can tweak the greedy heuristic to avoid some of these situations; at each step, we compute the *benefit* of a decision taking into account all prior decisions. The benefit is measured by data graph coverage, the metric we seek to maximize. At each stage, we greedily pick the decision with largest benefit. We present below this *weighted greedy* heuristic.

Algorithm. WeightedGreedy

```
1. Process labels in $L$ "bottom-up," so that label $X$ is processed after all labels $Y \in L$ with
   $X \in ancestor(Y)$.
2. Suppose we are currently processing label $X$.
     (i) For each label $Y \in ancestor(X)$:
          a. Set $H$ to be the empty graph.
          b. Traverse $G$ and visit each arc $(u, v)$ such that $label(v) = X$ and $label(u) = Y$.
          c. Let $G'$ be the subgraph of $G$ reachable from node $v$.
          d. Set $H = H \cup G'$.
          e. Set $benefit(X, Y) = |V_H|$, where $|V_H|$ is number of nodes in $H$.
    (ii) Let $Z \in ancestor(X)$ be the label with largest value of $benefit(X, Z)$.
   (iii) Set $parent(X) = Z$.
    (iv) Traverse $G$ and delete all edges of the form $(u, v)$ where $label(v) = X$ and $label(u) \neq Z$.
         Recursively, delete all nodes and edges that are disconnected from the root of $G$ by this
         deletion.
```

Example 6.2. For the data graph G in Fig. 13, we process the labels in the order $DCBAR$. We compute the following non-zero benefits:

- $benefit(D, C) = 4$, and $parent(D) = C$.
- $benefit(C, A) = 2$, $benefit(C, B) = 5$, so $parent(C) = B$.

- *benefit*$(B, R) = 6$, and *parent*$(B) = R$.
- *benefit*$(A, R) = 1$, and *parent*$(A) = R$.

The resulting skeleton is the best-fit skeleton with coverage 8.

6.4. Theoretical performance of heuristics

How badly can our heuristics perform, as a ratio of the optimal skeleton's coverage (called the *competitive ratio*)? We can modify the data graph in Fig. 13 to create data graphs where the greedy algorithm's competitive ratio is arbitrarily close to zero; that is, the greedy heuristic can perform arbitrarily badly compared to the optimal result. The weighted greedy heuristic works well for the data graph in Fig. 13, but we can construct data graphs where it does not pick the optimal skeleton. Indeed, we should be surprised if this were not the case; the best-fit problem is NP-complete, while the weighted greedy is a simple polynomial-time algorithm.

Fig. 14 shows a data graph where the weighted greedy algorithm picks a solution that has 0.625 the coverage of the optimal skeleton. We can modify this example to construct data graphs where the competitive ratio of weighted greedy gets arbitrarily close to 0.5. Using data graphs of greater depth, we can construct examples where the competitive ratio is arbitrarily close to zero: for each depth $d \geqslant 2$, we can construct an example where the competitive ratio approaches $(\frac{1}{2})^{d-1}$. In practice, however, we observe that the weighted greedy does much better than the theoretical worst-case bound, as shown by the experimental results of the following section.

6.5. Experimental results

For our experiments, we chose as our application a system that integrates job listings from multiple corporate websites into a single database. We chose a relation schema with a dozen attributes such as Job Id, Job Title, Job Category, Division, Location, and Job Description. Not all websites include information on all these attributes. We developed regular expression patterns that could identify these patterns on a small set of websites, including those of IBM [22] and Sun Microsystems [33] (a real system such as [20] or [35] incorporates more extensive heuristics to identify attributes).

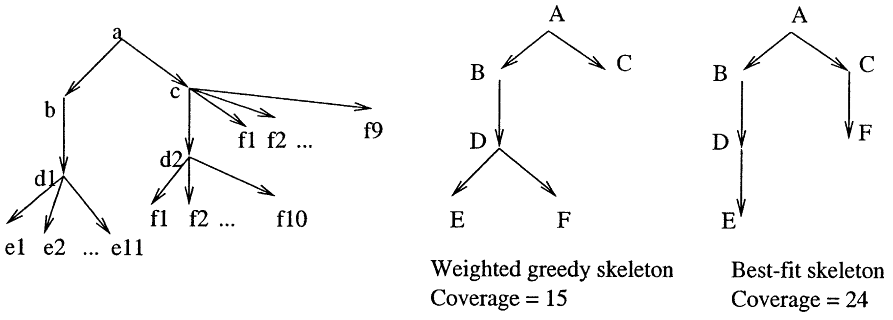


Fig. 14. A data graph where weighted greedy does badly.

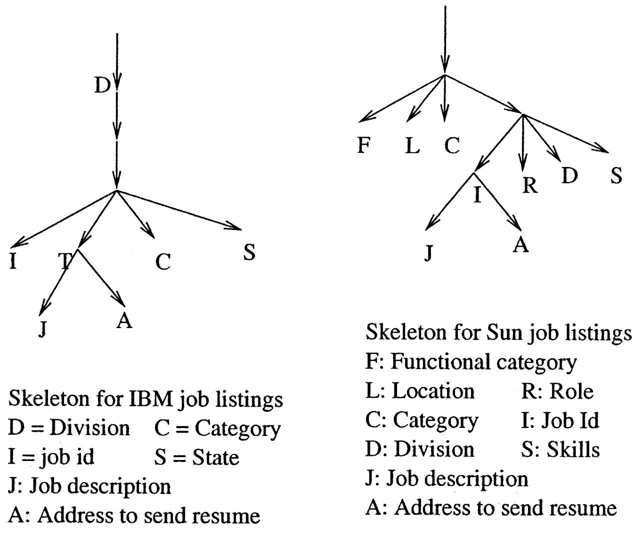


Fig. 15. Compact skeletons for portions of IBM and Sun websites listing job opportunities.

The data graphs we constructed had between 5000 and 10,000 nodes. Fig. 15 shows the skeletons generated for the IBM and Sun websites. Notice that the Sun website has information on many more attributes than the IBM website. The Sun website uses a form that is filled in at the top level; we constructed the data graph for this site using the technique for forms outlined in Section 7.2. The skeletons were constructed using the greedy and weighted greedy heuristics, which both constructed the same skeleton. We also verified manually that these skeletons are the optimal (best-fit) skeletons for these data graphs.

To study systematically how the greedy and weighted greedy heuristics performed as the noise level in the data graph is increased, we came up with a technique to randomly mutate the data graphs. We picked a single parameter, the *error rate* p, to model both the *precision* and the *recall* of the pattern matching functions: p is both the probability of a false positive and the probability of a false negative. We randomly mutated our data graphs assuming different values of p ranging from 0 (perfect precision and recall) to 1.0 (random data graphs), as follows. Consider the pattern-matching function that identifies instances of any attribute A in the data graph. In a mutated data graph with error rate p, this function incorrectly identifies instances of other attributes as being instances of A with probability p (i.e., has precision $1 - p$), and also misses actual instances of the attribute A with probability p (i.e., has recall $1 - p$).

In each case, we ran the greedy and weighted greedy heuristics on the mutated data graph, and we also ran an exhaustive enumeration algorithm that computes a best-fit skeleton.

Figs. 16 and 17 summarize the results of our experiments. We note that both heuristics compute the best-fit skeleton, for error rates below 0.2. For error rates larger than 0.2, the greedy heuristic decays linearly (we show a least-squares fit). Weighted greedy is always competitive to with a factor of 0.95 of the optimal, and thus is a very good heuristic in practice. There is a big payoff for the additional complexity of using weighted greedy versus the greedy heuristic, while there is very little payoff in using the much more expensive exhaustive search algorithm to compute the optimal skeleton.

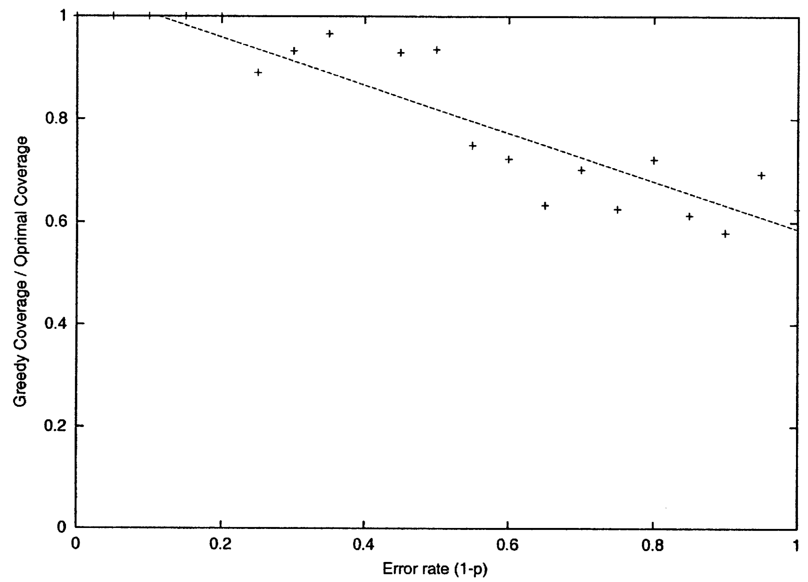


Fig. 16. Performance of greedy heuristic.

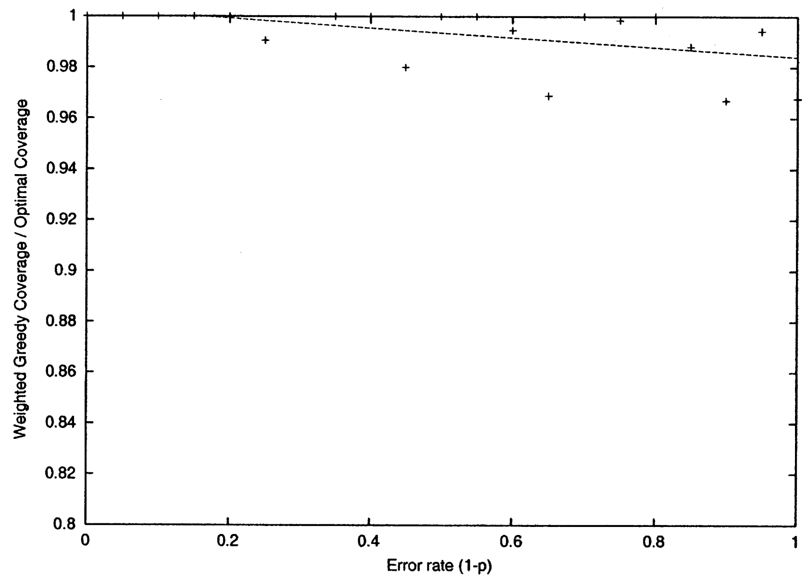


Fig. 17. Performance of weighted greedy heuristic.

7. Practical considerations

In this section we discuss some of the practical considerations that arise when applying the compact skeleton technique to construct wrappers for websites. We start with a discussion of how to map websites to data graphs, then consider websites with forms.

7.1. Structure within web pages

To model the structure of a website, data graphs must incorporate not only the links between pages, but also the structure within each page. The simplest approach is to model each web page as a single unlabeled node in the data graph. Each data element matched within the page is a child node of this node. There are also links between unlabeled nodes corresponding to links between webpages.

This simple model can be improved upon in several ways. It is possible to recognize structuring elements within the HTML of web pages: for example, list elements, header elements, and so on. When we encounter several data elements within a list, for instance, we might have a single unlabeled node in the data graph corresponding to the list, with the data elements in the list as its children. Constructing the data graph in this manner preserves more of the structure of the web page. Wrapper construction systems such as Junglee's VDBMS [20] incorporate such heuristics to construct data graphs that preserve as much of the website's structure as possible.

A common occurrence is data elements that always occur together in close proximity, for example, city, state, and zipcode. In such cases, there is often a single pattern that extracts multiple data elements e.g., a regular expression that matches patterns of the form "Beverly Hills, CA 90210." With patterns of this sort, the data graph can contain an unlabeled node corresponding to the entire pattern, whose children are the data values that were matched.

7.2. Forms

Websites often have forms that when filled and submitted result in a web page whose contents depend on the form values. Many websites with forms can be fit into the data graph model. For example, in many cases a form input is a menu or a drop-down list that contains data values. In other cases it is a radio button or a check box. In such cases, it possible by an automated procedure to fill the form using all possible permutations of values. Such a form can be modeled as follows: Create an unlabeled "form node" corresponding to each permutation of values. The set of data values for this particular permutation of form inputs are children of the form node. The root of the data graph corresponding (recursively) to the website whose root is the page returned by the form submission is a sibling to the form data values.

We have successfully modeled several websites with forms in this manner. The only condition is that the form have no "free-form" text inputs, because then its input permutations cannot be enumerated exhaustively.

8. Graph skeletons

Thus far we have restricted our attention to skeletons that are trees. There are websites that can be generated using more general skeletons, such as DAGs or even cyclic graphs. It turns out that theory of perfect compact skeletons generalizes to such cases: in particular, Theorem 2.1 generalizes so that every data graph has either a unique PCS or no PCS, even when the PCS is a cyclic graph. We provide below an algorithm to construct a PCS for a data graph G if it has one, and a proof that the PCS is unique. The algorithm is exponential in the size of G and it is unlikely that we can do better: determining whether a data graph has a DAG PCS turns out to be an NP-complete problem.

8.1. Distinguishing patterns

Given a data graph G, any PCS K induces a natural equivalence relation $\equiv_K$ on the nodes of G. For $u, v \in V_G$, we define $u \equiv_K G$ if $image_K(u) = image_k(v)$. We extend this equivalence relation to the arcs of G in the natural manner: for arcs $e_1 = (u_1, v_1)$ and $e_2 = (u_2, v_2)$, $e_1 \equiv_K e_2$ iff $u_1 \equiv_K u_2$ and $v_1 \equiv_K v_2$.

Let G be a data graph with PCS K. Our algorithm works by constructing $\equiv_K$, the equivalence relation on the nodes and arcs of G induced by K. The following lemma summarizes some straightforward observations.

Lemma 8.1. *Let K be a PCS for a data graph G, and let u and v be distinct nodes of G. Then the following statements are true*:

- *If $attr(u) = attr(v) \neq \perp$, then $u \equiv_K v$.*
- *If $u \equiv_K v$, then $attr(u) = attr(v)$.*
- *If $(u, v) \in E_G$ or $(v, u) \in E_G$, then $u \not\equiv_K v$.*
- *If u is the root of G, then $u \not\equiv_K v$.*

Let K be a PCS for G. A *K-pattern* is a subgraph P of G such that each node of P belongs to a different equivalence class with respect to K. Patterns P_1 and P_2 are *isomorphic* if there is a 1–1 mapping ϕ between their nodes such that $\phi(u) = v$ iff $u \equiv_K v$ and ϕ is an isomorphism in the graph-theoretic sense. Isomorphism between a K-pattern P and a subgraph K' of K is defined in the analogous manner. The following two lemmas are about K-patterns.

Lemma 8.2. *Let K be a PCS for a data graph G, and let P be a K-pattern of G. Then P is isomorphic to a subgraph of K.*

Proof. Follows from the observation that every edge of G belongs to some equivalence class under K, and every edge in a K-pattern belongs to a different equivalence class. □

Lemma 8.3. *Let $e_1, e_2 \in E_G$ such that $e_1 \equiv_K e_2$, and let P be any K-pattern that includes e_1. Then there is a K pattern P' isomorphic to P that includes e_2.*

Proof. From the previous lemma P is isomorphic to some subgraph K' of K. Since K is a PCS, there is a pattern Q that includes e_2 and is isomorphic to K. Therefore there is some subgraph P' of Q that is isomorphic to K'. □

8.2. The candidate skeleton

Given a data graph G, we present an algorithm to compute a *candidate skeleton* K corresponding to G. The candidate skeleton has the property that if G has a PCS, then the candidate skeleton is a PCS for G. It may happen that G has no PCS but we are still able to compute the candidate skeleton.

Our algorithm works by computing the equivalence relation $\equiv_K$. We use a symmetric boolean matrix M to encode this information, with $M_{ij} = true$ if $i \equiv_K j$. Initially, we set M_{ij} to *true* for all pairs of nodes except those mandated by Lemma 8.1. At each successive step, we use Lemma 8.3 to distinguish an additional pair of nodes in the following manner. At any stage in the algorithm, suppose nodes u, v are such that $M_{uv} = true$. Nodes u and v are *distinguishable* if one of the following conditions holds:

- There is an M-pattern P that includes u such that there is no isomorphic M-pattern P' that includes v.
- There is a node w with $(u, w) \in E_G$, $(v, w) \in E_G$, and there is an M-pattern P that includes (u, w) but not (v, w).
- There is a node w with $(w, u) \in E_G$, $(w, v) \in E_G$, and there is an M-pattern P that includes (w, u) but not (w, v).

Distinguishability is symmetric, so u is distinguishable from v iff v is distinguishable from u by the symmetric conditions to those listed above.

When we can distinguish nodes u and v, we set M_{uv} to *false*. If at any stage, we are able to distinguish a pair of nodes that must necessarily be equivalent (i.e., nodes with the same non-null attribute), then G has no PCS. Otherwise, when the algorithm terminates, we will have computed the relation $\equiv_K$. We show the full algorithm below. Since M is symmetric, whenever we assign M_{ij} in the algorithm, it is to be understood that we assign M_{ji} the same value.

Algorithm. CandidatePCS

```
1. Initialize matrix $M$ as follows:
     (i) If $r$ is the root of $G$, $M_{rs} = false$ for all nodes $s \neq r$.
    (ii) If $u$ and $v$ are nodes such that $attr(u) \neq attr(v)$, $M_{uv} = false$.
   (iii) If $(u, v) \in E_G$, then $M_{uv} = false$.
    (iv) $M_{ij} = true$ for all other pairs of nodes $i, j \in V_G$.
2. Repeat until there are no changes to $M$:
     (i) Find a pair of nodes $u, v$ such that $u$ and $v$ are distinguishable.
    (ii) Set $M_{ij}$ to $false$.
   (iii) If $attr(i) = attr(j) \neq \perp$, $G$ has no PCS.
3. $M$ is the equivalence relation corresponding to the candidate skeleton for $G$.
```

Lemma 8.4. *If u and v are nodes of G such that M_{uv} is set to false in Algorithm CandidatePCS, then there is no PCS K of G such that $u \equiv_K v$.*

Proof. We use induction on the number of iterations of the loop in Algorithm CandidatePCS. For the basis, Lemma 8.1 implies that the lemma is true for the pairs of nodes for which we set M_{uv} to *false* before any iteration of the loop. Assume that the lemma holds after $k-1$ iterations, and suppose we set M_{uv} to *false* in the kth iteration. Then Lemma 8.3 gives us the desired result. □

Let us say that Algorithm CandidatePCS terminates successfully if it gets to Step 3. In this case, it is straightforward to see that M is an equivalence relation: M is reflexive and symmetric by definition, and the definition of distinguishability ensures transitivity. We construct the skeleton K corresponding to M as follows. There is a node corresponding to each equivalence class of M. Let u and v be nodes of K. There is an arc (u, v) in K whenever there are nodes $x, y \in V_G$ with $u = image_K(x)$, $v = image_K(y)$, and $(x, y) \in E_G$. In addition, node u in K is labeled with attribute A whenever there is a node $x \in V_G$ with $u = image_K(x)$ and $attr(x) = A$. The labeling is consistent, because if the algorithm terminates successfully then all nodes in the same equivalence class have the same label.

To test whether K is a PCS for G, we test if for every edge $e \in E_G$, there is a K-pattern that contains e and is isomorphic to K. We now prove that G has a PCS if and only if K is a PCS for G.

Lemma 8.5. *If G has a PCS, then K is a PCS for G.*

Proof. Suppose K is not a PCS for G, and G has a PCS L. From Lemma 8.4, it follows that $\equiv_L$ is a strict refinement of $\equiv_K$. By induction on the process of refinement to get from $\equiv_K$ to $\equiv_L$, we can show that K is a subgraph of L. Let $e \in V_G$; since L is a PCS, e is part of an L-pattern P. Since K is a subgraph of L, there is a subpattern P' of P that is isomorphic to K. We consider two cases:

Case 1: e is in P'; in this case there is a K-pattern in G containing e.

Case 2: e is not in P'. Because $\equiv_L$ is a refinement of $\equiv_K$, there is some edge e' satisfying Case 1 such that $e \equiv_K e'$. Since e' is part of a K-pattern, by construction of $\equiv_K$ it follows that e is also part of a K-pattern.

Thus in either case e is part of a K-pattern. Since e is an arbitrary edge of G, we have shown that every edge of G is part of some K-pattern, and so K is a PCS for G. □

Lemma 8.6. *If K is a PCS for G and L is a skeleton such that $\equiv_L$ is a strict refinement of $\equiv_K$, then L is not a PCS for G.*

Proof. If K is a PCS for G and $\equiv_L$ is a strict refinement of $\equiv_K$, then we can show by induction on the refinement process that K is a PCS for L when L is treated as a data graph (replacing each attribute label with a value corresponding to that attribute). Let $\sigma_1, \ldots, \sigma_n$ be all possible overlays of K on L.

Suppose L is a PCS for G. Consider some overlay ϕ of L on G, corresponding to an L-pattern P. Construct an overlay ϕ' of L on G corresponding to the same L-pattern P as follows:

- For each node u of L such that u is in σ_1, let v be the node of L in σ_2 such $u \equiv_K v$. If $\phi(u) = w$, set $\phi(v) = w$.
- For each node v of L such that v is in σ_2, let u be the node of L in σ_1 such that $u \equiv_K v$. If $\phi(v) = w$, set $\phi(u) = w$.
- For all other nodes of L, $\phi = \phi'$.

Now ϕ and ϕ' are overlays of L that map some node w of G to different nodes of L, implying that L is not compact for G. $\square$

We define a skeleton K to be *irreducible* if there is no skeleton K' that is a PCS for K when K is treated as a data graph. From the proof of Lemma 8.6, it follows that only irreducible skeletons can be compact. Lemmas 8.4–8.6 imply the following theorem, which is the counterpart of Theorem 2.1.

Theorem 8.1. *For any relation R, data graph G, and irreducible skeleton K, if $R \xrightarrow{K} G$, then K is the unique PCS for G. Conversely, for any data graph G with PCS K, there is a relation R such that $R \xrightarrow{K} G$.*

8.3. Complexity

Algorithm CandidatePCS is exponential in the size of the data graph G, as is the subsequent test to check whether the candidate skeleton is a PCS. The following theorem shows that we cannot hope to find a polynomial-time algorithm for the PCS problem, even for DAG-structured data graphs with an information element at each node.

Theorem 8.2. *The problem of determining whether a DAG-structured data graph has a DAG PCS is NP-complete.*

Proof. Let G be a data graph structured as a DAG with an information element at each node. To see that the problem is in NP, note that we can guess in polynomial time a DAG PCS K and for every edge in G an overlay that covers that edge. It follows that K must be compact because each node u of G contains an information element and every overlay must map it to the node of K labeled with $attr(u)$.

To show that the problem is NP-hard, we provide a reduction from the problem of determining whether the natural join of several relations is non-empty, which is known to be NP-complete [36]. It can be verified that this problem remains NP-complete when it is constrained so that among the relations $R_1, \ldots, R_n$ there is a pair of join-consistent relations (without loss of generality, R_1 and R_2) that each contain a single tuple.

Given such a collection of relations, create an instance of the DAG PCS problem as follows. There is an attribute A_i corresponding to each relation R_i. Construct graph G as follows. For each tuple $t \in R_i$, create a node with some unique value a and include a in $Dom(A_i)$. Whenever tuples $t_i \in R_i$ and $t_j \in R_j$ with corresponding node values a_i and a_j and $i < j$ are join-consistent, add a directed edge from a_i to a_j. Note that there will be exactly one node labeled with a value corresponding to A_1 and one node with a value corresponding to A_2, with an edge e between them.

Construct skeleton K as follows. There is a node u_i labeled with A_i for each attribute name A_i. Whenever R_i and R_j share an attribute and $i < j$, add a directed edge from u_i to u_j. Now add some additional nodes and edges to G as follows.

For each edge $(a_i, a_j) \in E_G$ with $(a_i, a_j) \neq e$, create a copy K_{ij} of K with the attributes A_i and A_j replaced by values a_i and a_j respectively; all other attributes are replaced by unique data values in the domains of their corresponding attributes that do not appear elsewhere in G. Augment G by adding K_{ij} and identifying the nodes with values a_i and a_j in G with the corresponding nodes in K. Repeat this process for every edge in G except for e.

It can be verified that G and K are both DAGs since all edges go from lower to higher numbered nodes, and that the sizes of G and K are polynomial in the sizes of the relations. We claim that $R_1 \bowtie \cdots \bowtie R_n$ is nonempty if and only if K is a PCS for G.

IF: Suppose K is a PCS for G. Then there is an overlay of K on G that includes e. This overlay includes a data value a_i corresponding to each attribute A_i, $1 \leqslant i \leqslant n$. Let t_i be the tuple in relation R_i corresponding to the data value a_i, $1 \leqslant i \leqslant n$. Due to the manner in which G was constructed, each pair of tuples (t_i, t_j) that share an attribute agree on their common attributes (else there would not be an edge between a_i and a_j in G). Thus the set of tuples $t_1, \ldots, t_n$ is join consistent and so $R_1 \bowtie \cdots \bowtie R_n$ is non-empty.

ONLY IF: Suppose $R_1 \bowtie \cdots \bowtie R_n$ is non-empty. We show that there are overlays of K on G that include every edge e' of G. There are two cases:

Case 1: $e' \neq e$. Then e' is in one of the subgraphs K_{ij} isomorphic to K that was added to G in the last step of the construction. Therefore there is an overlay of K that includes e (the corresponding K-pattern is the subgraph K_{ij}).

Case 2: $e' = e$. Since the $R_1 \bowtie \cdots \bowtie R_n \neq \emptyset$, there are tuples $t_i \in R_i$, $1 \leqslant i \leqslant n$, that are join consistent. The subgraph H of G induced by the nodes corresponding to these tuples is a K-pattern that includes e.

Thus every edge of G is included in some overlay of K, and so K is a PCS for G. □

9. Related work

Integrating data across websites is an active area of research, with several research prototypes and commercial implementations [12,15,20,25,35]. Constructing wrappers has been the major bottleneck in most of these systems. Hammer et al. [21], one of the earliest systems, describes a toolkit that helps the user manually code wrappers. The toolkit provides many constructs, especially for HTML processing, that make it easier in many cases than writing parsers using Lex and Yacc.

There has been much work in automating aspects of wrapper construction [2–5,13,14,20,23,24,27,32]. Most of this work has focused on extracting document structure (as a grammar or a finite state automaton) from HTML and text documents. Adelberg [2] and Garofalakis et al. [17] describe systems that can infer structure from HTML and XML documents, respectively, in a semiautomated manner. Ashish and Knoblock [3,4] present a technique called *wrapper induction* that can infer simple grammars that combine HTML elements and data elements on web pages. Kushmerick et al. [24], Muslea et al. [27] and Soderland [32] describe machine learning approaches. In contrast to these works, which infer structure within web pages, compact skeletons describe web site structure within and across web pages, and in addition also enable automatic transformation between relational and web data.

Gupta et al. [19] describes a system in which wrappers are expressed in a specialized language called *site description language* (SDL). It can be shown that SDL corresponds to a restricted form of canonical skeletons we call *skinny skeletons*: skeletons that are trees consisting of a single root-to-leaf *distinguished path* such that every node is at distance at most 1 from this path. The VDBMS system described in [20] has a wrapper toolkit that uses an algorithm similar to Algorithm *ComputeRelation* in order to compute the relation corresponding to a website. This algorithm is simpler than Algorithm *ComputeRelation* because the skeleton is restricted to be skinny, and has the advantage that it can incrementally produce the tuples of the relation as it traverses the pages of the website.

Although many data graphs cannot be modeled using skinny skeletons, such skeletons do appear in practice to capture the structure of many useful websites; for example, both the IBM and Sun skeletons shown in Fig. 15 are skinny skeletons. However, an important restriction of the VDBMS system described above is that the SDL description for a website needs to be manually generated, whereas this paper describes algorithms to automatically deduce the compact skeleton from the website.

Jensen and Cohen [23] propose a language to describe how fields extracted from different portions of a website are to be grouped together into records. This language describes hierarchical structures in a manner similar to skeletons. In the system described in [23], the user can create and modify different groupings and compare the results using a GUI tool, and manually pick the best grouping.

Embley and Xu [13] propose a different approach to the problem discussed in this paper. They study a technique based on the Vector Space Model, a common information-retrieval measure of document relevance. They construct an ontology that describes a domain of interest (e.g., used car ads), and then consider how to rearrange the data on a website so as to maximize the vector space model measure relative to the ontology. In contrast, our approach uses simple graph-theoretic models and measures. It is our intuition that the skeleton approach is more readily applicable to large programmatically generated websites, while the vector space model approach works better on websites containing a substantial quantity of human-generated textual data.

There has been much work on querying documents, semistructured data, and unstructured data [1,6,8–11,26,30,31]. Many of these systems model semistructured data using data graphs. Some consider schemas that are similar in spirit to compact skeletons. The focus of these works is on query languages, testing whether a database conforms to a given schema, schema subsumption, query optimization, and data translation, in contrast to our focus on schema inference.

Nestorov et al. [28] presents a different approach to extract structure from semistructured data. They model semistructured data using data graphs and construct a *typing* that fits the data within an error threshold. Typings are defined in terms on monadic datalog programs, and nodes in the data graph can play multiple "roles." Compact skeletons by contrast provide a simple graph-theoretic approach to schema inference that works well for data on the web, and lend themselves to simple algorithms that perform well on large websites.

Representative Objects [29] and Data Guides [18] provide structural summaries of hierarchical semistructured data in a manner similar to skeletons. The purpose of data guides is to facilitate browsing and query optimization rather than mapping data into the relational model. Therefore, the data guide model allows more general structures than skeletons: data guides are not restricted to be trees and the same label can appear more than once in a data guide. The different intended applications lead to problems and algorithms of a different flavor from those presented here.

10. Conclusion

Compact skeletons are a simple and effective model for "reverse-engineering" websites to construct wrappers that expose relational interfaces. The model can equally well be applied to XML documents where the DTD is not specified in advance. We are exploring several extensions to the basic model:

- Feedback between feature (i.e., data element) extraction and structure extraction.
- Using domain knowledge about the relation e.g., functional dependencies, to refine our heuristics for skeleton construction.
- Data graphs corresponding to multiple compact skeletons. In such cases, we must construct a "small" set of skeletons that cover as much of the data graph as possible, trading off between coverage and the number of skeletons.

Acknowledgments

The authors thank the anonymous referees for pointing out some related work and for other helpful suggestions.

References

[1] S. Abiteboul, Querying semistructured data, in: Proceedings on the International Conference on Database Theory, 1997.

[2] B. Adelberg, Nodose—a tool for semi-automatically extracting structured and semistructured data from text documents, in: Proceedings of the ACM SIGMOD International Conference of Management of Data, 1998.

[3] N. Ashish, C. Knoblock, Semi-automatic wrapper generation for internet information sources, in: Proceedings of CoopIS '97, 1997.

[4] N. Ashish, C. Knoblock, Wrapper generation for semi-structured internet sources, SIGMOD Rec. 26 (4) (1997) 8–15.

[5] P. Atzeni, G. Mecca, Cut and paste, in: Proceedings of the Sixteenth ACM Symposium on Principles of Database Systems, 1997, pp. 144–153.

[6] C. Beeri, T. Milo, Schemas for integration and translation of structured and semistructured data, in: Proceedings of the International Conference on Database Theory, 1999.

[7] S. Brin, Extracting patterns and relations from the world-wide web, in: International WebDB Workshop, Valencia, Spain, 1998, pp. 172–183.

[8] P. Buneman, S. Davidson, M. Fernandez, D. Suciu, Adding structure to unstructured data, in: Proceedings on the International Conference on Database Theory, 1997.

[9] P. Buneman, S. Davidson, G. Hillebrand, D. Suciu, A query language and optimization techniques for unstructured data, in: Proceedings of the ACM SIGMOD International Conference of Management of Data, 1996.
[10] V. Christophides, S. Abiteboul, S. Cluet, M. Scholl, From structured documents to novel query facilities, in: Proceedings of the ACM SIGMOD International Conference of Management of Data, 1994.
[11] S. Cluet, C. Delobel, J. Simeon, K. Smaga, Your mediators need data conversion! in: Proceedings of the ACM SIGMOD International Conference of Management of Data, 1998.
[12] R. Doorenbos, O. Etzioni, D.S. Weld, A scalable comparison-shopping agent for the world-wide web, in: Proceedings of the First International Conference on Autonomous Agents, 1997.
[13] D.W. Embley, D.M. Campbell, Y.S. Jiang, Y.-K. Ng, R.D. Smith, S.W. Liddle, D.W. Quass, A conceptual-modeling approach to extracting data from the web, in: Proceedings of the 17th International Conference on Conceptual Modeling (ER '98), 1998.
[14] D.W. Embley, L. Xu, Record location and reconfiguration in unstructured multiple-record web documents, in: JJCAI-2001 Workshop on Adaptive Text Extraction and Mining, 2001.
[15] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J.D. Ullman, V. Vassalos, J. Widom, The TSIMMIS approach to mediation: data models and languages, J. Intell. Inform. Systems 8 (2) (1997) 117–132.
[16] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman and Company, New York, NY, 1979.
[17] M. Garofalakis, A. Gionis, R. Rastogi, S. Seshadri, K. Shim, XTRACT: a system for extracting document type descriptors from XML documents, in: Proceedings of the ACM SIGMOD International Conference of Management of Data, 2000.
[18] R. Goldman, J. Widom, Data guides: enabling query formulation and optimization in semistructured databases, in: Proceedings of the 23rd International Conference on Very Large Data Bases, 1997.
[19] A. Gupta, V. Harinarayan, D. Quass, A. Rajaraman, Method and apparatus for structuring the querying and interpretation of semistructured information, United States Patent number 5,826,258, 1998.
[20] A. Gupta, V. Harinarayan, A. Rajaraman, Virtual database technology, in: Proceedings of the Fourteenth International Conference on Data Engineering, February 23–27, 1998, Orlando, Florida, USA, IEEE Computer Society, Silver Spring, MD, 1998, pp. 297–301.
[21] J. Hammer, H. Garcia-Molina, J. Cho, R. Aranha, A. Crespo, Extracting semistructured information from the web, in: Workshop on management of semistructured data, 1997.
[22] IBM Corp, Job listings at IBM corporate website, `http://www.ibm.com/employment/us/html/location.html`.
[23] L. Jensen, W. Cohen, Grouping extracted fields, in: WebDB (Informal Proceedings), 2000, pp. 123–128.
[24] N. Kushmerick, D.S. Weld, R. Doorenbos, Wrapper induction for information extraction, in: Proceedings of the 1997 International Joint Conference on Artificial Intelligence, 1997.
[25] A.Y. Levy, A. Rajaraman, J.J. Ordille, Querying heterogeneous information sources using source descriptions, in: Proceedings of the 22nd International Conference on Very Large Data Bases, 1996, pp. 251–262.
[26] T. Milo, S. Zohar, Using schema matching to simplify heterogeneous data translation, in: Proceedings of the 24th International Conference on Very Large Data Bases, 1998.
[27] I. Muslea, S. Minton, C. Knoblock, Stalker: learning extraction rules for semistructured, web-based information sources, in: Proceedings of AAAI '98: Workshop on AI and Information Integration, 1998.
[28] S. Nestorov, S. Abiteboul, R. Motwani, Extracting schema from semistructured data, in: Proceedings of the ACM SIGMOD International Conference of Management of Data, 1998.
[29] S. Nestorov, J. Ullman, J. Wiener, S. Chawathe, Representative objects: coincide representations of semistructured hierarchical data, in: Proceedings of the 13th International Conference on Data Engineering, 1997.
[30] D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, J. Widom, Querying semistructured, heterogeneous information, in: Proceedings of the Fourth International Conference on Deductive and Object Oriented Databases, 1995.

[31] J. Shanmugasundaram, K. Tufte, G. He, C. Zhang, D. DeWitt, J. Naughton, Relational databases for querying XML documents: limitations and opportunities, in: Proceedings of the 25th International Conference on Very Large Data Bases, 1999.
[32] S. Soderland, Learning to extract text-based information from the world-wide web, in: proceedings of the Third International Conference on Knowledge Discovery and Data Mining, 1997.
[33] Sun Microsystems, Job listings at Sun Microsystems website. `http://vww.sun.com/jobs`.
[34] WhizBang! Labs, Flipdog.com job search website. `http://www.flipdog.com/home.html`.
[35] WhizBang! Labs, WhizBang! Labs corporate website. `http://wwv.whizbanglabs.com`.
[36] M. Yannakakis, Algorithms for acyclic database schemes, in: Proceedings of the Seventh International Conference on Very Large Data Bases, 1981, pp. 82–94.