Abstra: Toward Generic Abstractions for Data of Any Model
Nelly Barret, Ioana Manolescu, Prajna Upadhyay

To cite this version:
Nelly Barret, Ioana Manolescu, Prajna Upadhyay. Abstra: Toward Generic Abstractions for Data of Any Model. CIKM 2022 - 31st ACM International Conference on Information and Knowledge Management, Oct 2022, Atlanta, Georgia / Hybrid, United States. hal-03767967

HAL Id: hal-03767967
https://inria.hal.science/hal-03767967
Submitted on 2 Sep 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
ABSTRACT

Digital data sharing leads to unprecedented opportunities to develop data-driven systems for supporting economic activities, the social and political life, and science. Many open-access datasets are RDF (Linked Data) graphs, but others are JSON or XML documents, CSV files, Neo4J property graphs, etc. Potential users need to understand a dataset in order to decide if it is useful for their goal. While some published datasets come with a schema and/or documentation, this is not always the case. We demonstrate ABSTRA, a dataset abstraction system, which applies on a large variety of data models. ABSTRA computes a description meant for humans, and integrates Information Extraction to classify dataset content among a set of categories of interest to the user. Our abstractions are conceptually close to Entity-Relationship diagrams, but our entities can have deeply nested structure.

1 INTRODUCTION

Open-access data being shared over the Internet enabled the development of new businesses, economic opportunities and applications; it also leads to circulating knowledge on health, education, environment, the arts, science, news, etc. The World Wide Web Consortium’s recommended data sharing format is as RDF (Linked data) graphs. However, in practice, other formats are also widely used. For instance, bibliographic notices on PubMed, a leading medical scientific site, are available in XML; JSON is increasingly used, e.g., on social networks; CSV files are shared on portals such as Kaggle. Relational databases are sometimes shared as dumps, including schema constraints such as primary and foreign keys, or as CSV files; property graphs [3] (PGs, in short, such as pioneered by Neo4J) are used to share Offshore leaks, a journalistic database of offshore companies, etc.

Users who must decide whether to use a dataset in an application need a basic understanding of its content and the suitability to their need. Towards this goal, schemas may be available to describe the data structure, yet they have some limitations. (i) Schemas are often unavailable for semistructured datasets (XML, JSON, RDF, PGs). Even when a schema is supplied with or extracted from the data, e.g., [5, 10, 19, 23]. (ii) Schema syntactic details, such as regular expressions, etc., are hard to interpret for non-expert users. (iii) A schema focuses primarily on the dataset structure, not on its content. It does not exploit the linguistic information encoded in node names, in the string values the dataset may contain, etc. (iv) Schemas employ the data producer’s terminology, not the concepts of interest to users. (v) Schemas do not quantitatively reflect the dataset, whereas knowing “what is the main content of a dataset” can be very helpful for a first acquaintance with it. Data summaries can be built from semistructured data, e.g., [8, 12], but they may still be quite large, and they do not reflect user interest. An RDF dataset may come with an ontology describing its semantics, which is a step toward lifting limitation (iii); but, all the others still apply. Mining for patterns [15] allows to find popular motifs, e.g., items often purchased together. This avoids shortcomings (i) and (o), but not the others. Dataset documentation, when well-written, is most helpful. However, it still suffers from the issues (i) and (io) above: it is often lacking, and it reflects the producer’s view. In a data lake context, [14, 20] discovers, extracts, and summarizes structural metadata, and annotates data and metadata with semantic information. However, they focus on rooted or hierarchical data, while we also handle graphs (RDF or PGs) that may be cyclic, and our abstractions can also capture such cyclic relationships when present in the data.

We propose to demonstrate ABSTRA, a all-in-one system for abstracting any relational, CSV, XML, RDF or PG dataset. ABSTRA is based on the idea that any dataset comprises some records, typically grouped in collections (which we view as sets). Records describe entities or relationships in the classical conceptual database design sense [21]. ABSTRA entities can have deeply nested structure. When several collections of entities co-exist in a dataset, relationships typically connect them. ABSTRA proceeds as follows.

(1). Given any dataset, ABSTRA models it as a graph, and identifies collections of equivalent nodes, leveraging graph structural summarization, as we describe in Section 2.

(2). Among the collections, ABSTRA detects a few main collections, which together, hold a large part of the dataset contents. Each collection contains a set of similar, potentially deeply nested records. The challenge here is to detect, in the data graph, the nodes and edges that are “part of” each main collection record, and to do so efficiently even if the graph has complex, cyclic structure. This is addressed by introducing a notion of data weight and exploiting it as we describe in Section 3.

(3). ABSTRA attempts to classify each main collection into a given semantic category, such as Person, Product, GeographicPosition, etc., based on semantic resources resulting from prior work [16]. The classification also leverages Information Extraction to detect the presence of entities in the data values, as well as language models to detect proximity between the dataset vocabulary, and the target categories (Section 4).

ABSTRA outputs an Entity-Relationship style diagram of the main, classified collections, together with the possible relationships in which they participate; this description is free of any data model-specific details. For instance, given an XMark [22] XML document describing an online auctions site, containing 2.3M nodes with 80 different labels, ABSTRA returns: “A collection of Person entities, a collection of Product, and a collection of category” (the latter are used to describe the items for sale). Users can explore entities, and/or sample entity instances, through an interactive GUI (see Section 5).

Below, we describe the abstraction steps and outline the demonstration scenarios before concluding. ABSTRA examples and a video can be found at: https://team.inria.fr/cedar/projects/abstra/.
trees and RDF. Documents are modeled as trees. To the extent possible, authors
which we concatenate.

We first explain how any dataset is converted into a graph representa-
which connect a map parent to its children, on the child nodes, and
we attach meaningful, non-empty names to nodes, as illustrated in
XML, JSON and RDF. CSV files, relational databases and PGs are
used to denote the collection of text children of the nodes from the collection with the label year
and similarly for the others whose label end in #. The dw attributes
will be discussed in Section 3.

3 IDENTIFYING THE MAIN ENTITIES TO
REPORT AND THEIR RELATIONSHIPS

Among the collections C, some are more representative of the
dataset than others, e.g., in Figure 3, paper seems a better candidate
than its child collection year. However, we cannot select “the parent
(or root) collection(s)”, as the collection graph may lack a root node,
if it is cyclic as in Figure 3 (cycle edges are shown in red). Even if a
root collection exists, it may not be the best choice. For instance, consider XHTML search results grouped in pages, of the form (top)(page).../(result).../(result)(result).../(result)(page).../(page).../(top). Here, the top collection is that of pages, but the actual data is in the results, thus, “a collection of results” is a better abstraction.

### 3.1 Overview of the method

A high-level view of our method is the following (concrete details will be provided below):

1. **Selecting the main entities** (Section 3.2):
   (a) We assign to each collection a weight, and to each edge in the collection graph, a transfer factor.
   (b) We propagate weights in the collection graph, based on the weights and transfer factors, to assign to each collection a score that reflects not only its own weight, but also its position in the graph.
   (c) In a greedy fashion, we select the main entities by repeating:
      (i) Select the collection node \( C_E \) currently having the highest score, as a root of a main entity;
      (ii) Determine the boundary of the entity \( C_E \): this is a connected subgraph of the collection graph, containing \( C_E \). We consider all this subgraph as part of \( C_E \), which will be reported to users including all its boundary;
      (iii) Update the collection graph to reflect the selection of \( C_E \) and its boundaries, and recompute the collection scores; until a certain maximum number \( E_{\text{max}} \) of entities have been selected, or these entities together cover a sufficient fraction \( \text{cov}_{\text{min}} \) of the data.

2. **Selecting relationships between the main entity collections**. These relationships will also be reported as part of the abstraction (Section 3.3).

### 3.2 Main entity selection

We assign to each leaf node in \( G \) an own data weight \( (ow) \) equal to the number of edges incoming that node. In tree data formats, \( ow \) is 1; in RDF, for instance, a literal that is the value of many parameters: \( \text{E}_{\text{max}} \) and \( \text{cov}_{\text{min}} \) (Section 3.1), and \( f_{\text{min}} \) (Section 3.2). Empirically, we have used \( \text{E}_{\text{max}} \in \{3, 5\} \), \( \text{cov}_{\text{min}} = 0.8 \) and \( f_{\text{min}} = 0.3 \). More generally, classical Entity-Relationship (E-R) modeling is known to include a subjective factor, and for a given database, several E-R models may be correct. Our focus is on not missing any essential component of the dataset, while allowing users to limit the amount of information with \( \text{E}_{\text{max}} \), and classifying the main entities into categories, to make them as informative as possible (Section 4).

### 3.3 Relationship selection

Having selected the main entities \( \{C_{E1}^1, \ldots, C_{E}^{\text{max}}\} \) and their boundaries, every oriented path in the collection graph that goes from a given \( C_E^i \) to another \( C_E^j \) is reported as a relationship. For instance, in Figure 3, if the main entities are author (a), with mail and name in its boundary, and paper (p) with year, title and abstract in its boundary, the relationships are: \( \text{pr} \quad \text{w} \rightarrow \text{a}, \quad \text{h} \rightarrow \text{p}, \quad \text{pl} \rightarrow \text{c} \), and \( \text{inv} \rightarrow \text{a} \).

If the scores lead to reporting three main entities, the two above and also conf (c) with name and inv in its boundary, the relationships are:

- **Two weight propagation methods.**
  - We run the PageRank [7] algorithm on the collection graph with the edge direction inverted, so that each node transfers \( f_{ij} \) of its weight to its parent. Initially, only leaf collections have non-zero \( ow \), but successive PageRank iterations spread their weights across the graph. We call this method \( \text{PR}_{ow} \).
  - Our second method, denoted \( \text{prop}_{ow} \), propagates weights still backwards, but only outside of the collection graph cycles. Specifically, we assign to each collection a data weight \( dw \), which on leaf collection is initialized to \( ow \), and on others, to 0. Then, for each non-leaf \( C_i \), and non cyclic path from \( C_i \) to a leaf collection \( C_k \), we increase \( dw(C_i) \) by \( f_{ik} \cdot ow(C_k) \).

For instance, using the second method, the collection author in Figure 3 obtains \( dw = 6 \), corresponding to 3 transferred from mail, and 3 transferred from name. The intuition behind \( \text{prop}_{ow} \) is that edges that are part of cycles may have a meaning closer to “symmetric relationships between entities”, than to “including a collection in another collection’s boundary”. To determine entity boundaries, we proceed as follows:

- When using \( \text{prop}_{ow} \) we consider part of the boundary of an entity \( C_i \), any entity \( C_k \) that transferred some weight to \( C_i \), and all the edges along which such transfers took place. In Figure 3, mail and name are within the boundary of author.
- When using \( \text{PR}_{ow} \), to determine the boundary of \( C_i \), we traverse the graph edges starting from \( C_i \) and include its neighbor node \( C_j \) if and only if (i) the edge \( C_i \rightarrow C_j \) has a transfer factor of at least \( f_{ij} \), or (ii) each node from \( C_i \) has at most one child in \( C_j \). The intuition for (ii) is that \( C_j \) “can be assimilated to an attribute of \( C^*_i \), rather than being independent of it”.

Finally, to update the graph after selecting one main entity \( C_E \), each leaf collection in the boundary of \( C_E \) subtracts from its own weight \( ow \) the fraction (at most 1.0) that it propagated to \( C_E \). For instance, once author is selected with name in its boundary, the \( ow \) of name decreases to 2. Then, the scores of all graph collections (\( dw \), respectively, PageRank score based on \( ow \)) are recomputed.

### 3.4 Discussion

\text{Abstra} may return different results on a given dataset, depending on the scoring method used (\( \text{prop}_{ow} \) or \( \text{PR}_{ow} \)), as well as the parameters: \( \text{E}_{\text{max}} \) and \( \text{cov}_{\text{min}} \) (Section 3.1), and \( f_{\text{min}} \) (Section 3.2). Empirically, we have used \( \text{E}_{\text{max}} \in \{3, 5\} \), \( \text{cov}_{\text{min}} = 0.8 \) and \( f_{\text{min}} = 0.3 \). More generally, classical Entity-Relationship (E-R) modeling is known to include a subjective factor, and for a given database, several E-R models may be correct. Our focus is on not missing any essential component of the dataset, while allowing users to limit the amount of information with \( \text{E}_{\text{max}} \), and classifying the main entities into categories, to make them as informative as possible (Section 4).
To each main entity $C$ thus identified, we want to associate a category from a predefined set $K$ of semantic classes, and using also a set $P$ of semantic properties, which have known domain and range constraints connecting them to the classes in $K$.

We build our $K$ and $P$ by leveraging GitTables [16], a repository of 1.5M tables extracted from Github. For each attribute name encountered in a table, it provides candidate properties from DBpedia [4] and/or schema.org (https://schema.org/); it also provides the domain and range triples corresponding to these properties. For instance, GitTable’s entry for gender domain and range triples corresponding to these properties. For instance, GitTable’s entry for gender is:

"id": "schema:gender", "label": "gender", "range": "[schema:GenderType]",
"domain": ["schema:Person", "schema:SportsTeam"]

GitTables have been populated using SHERLOCK [17], a state-of-the-art deep learning semantic annotation technique. From GitTables, we derive 4.187 $P$ properties; 3,687 among them have domain information, and 3,898 have range statements.

The overall classification process is outlined in Figure 4; solid arrows connect associated data items and trace the classification process, while dotted arrows go from a set to one of its elements.

We exploit two kinds of information attached to $C$:

1. We consider each data property $dp_i$ that $C$ has, such as mail for the author collection in Figure 3. Out of all the values that $dp_i$ takes on a node from $C$, we compute an entity profile $\sigma_{C,dp_i}$, reflecting the entities extracted from these values. For instance, $\sigma_{author,mail}$ states that each value of the property mail contains an email (blue highlight in Figure 2), and that overall, 100% of the length of these values is part of an email entity. In general, a profile may reflect the presence of entities of several types, which may span over only a small part of the property values.

2. We compare $dp_i$ and $\sigma_{C,dp_i}$ to each property $p \in P$ and the classes in the range of $p$. If the property name $dp_i$ is sufficiently similar (through word embeddings) with some property $p \in P$, and that $\sigma_{C,dp_i}$ is similarly similar to a class $k \in K$, e.g., EmailAddress, that is in the range of $p$, this leads to a vote of $dp_i$ for classifying $C$, in every classes $k \in K$ such that $k$ is in the domain of $p$. Each property $dp_i$ may “vote” in favor of several classes, via different domain constraints; the higher the similarity between $dp_i$ and $p$, the more frequent $dp_i$ is on $C$ nodes, the fewer domain and range constraints $p$ has, the stronger the vote is.

The labels of $C$ nodes may also “vote” toward classifying collection $C$. All $C$ nodes may have the same label, e.g., author, which may resemble the name of a class, e.g., Author. Or, $C$ nodes may all have different names, e.g., RDF URLs of the form http://ns.com/Author123, from which we extract the component after the last /, eliminate all but alphabet letters, and use the result(s), weighted by their support among $C$ nodes, in a similar fashion.

Finally, $C$ is classified with the class $k^* \in K$ having received the highest sum of votes. The process resembles domain inference using RDF Schema ontology constraints, with the difference that our “votes” are quantified by similarity and support, and, to keep things simple for the users, we select a single class.

**5 SYSTEM AND SCENARIOS**

Abstra is implemented in Java, leveraging the graph creation (including entity extraction) and Postgres-based store of ConnectionLens [2]. All Abstra steps scale up linearly in the data size, which we experimentally verified on datasets of up to tens of millions of edges. The main memory needs are in TypedStrong partitioning, namely $O(|N|)$; other operations are implemented in SQL and benefit from Postgres’ optimizations.

We have computed abstractions of dozens of synthetic benchmark datasets used in the data management literature, such as XMark [22], BSBM [6], LUBM [13], and real-life datasets about cooking, NASA flights, clean energy production, Nobel prizes, CSV datasets from Kaggle, etc. varying the data model, the number of collections and entity complexity, the presence of relations, etc. During the demonstration, users will be able to: (i) change the system parameters (Section 3.4), and see the impact on the classification results; (ii) edit the semantic information $K$ and $P$ to influence the entity classification; (iii) edit a set of small RDF, JSON and XML examples, then abstract them to see the impact.

Abstractions are shown both as HTML text and as a light-weight E-R diagram of the main entity collections and their relationships (see Figure 5).

**6 CONCLUSION**

Abstra extracts the main entities and relationships from heterogeneous-structure datasets, leveraging Information Extraction, language...
Toward Generic Abstractions for Data of Any Model

models, knowledge bases, and user input to generate compact, easy-to-understand abstractions, and categorize them according to user’s interest. Abstra complements schema extraction [5, 10, 19, 23] or data profiling [1], aimed at more technical uses and users; Abstra aims to help novice, first-time users discover and start interacting with the data. More advanced visualization techniques then be used once, see, for instance, [18] for RDF graphs.

REFERENCES